A VISUAL LANGUAGE FOR COMPOSABLE SIMULATION SCENARIOS

THESIS

Carolyn R. Bartley, 2nd Lieutenant, USAF
AFIT/GCS/ENG/03-03

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/03-03

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U. S. Government.

AFIT/GCS/ENG/03-03

A VISUAL LANGUAGE FOR COMPOSABLE SIMULATION SCENARIOS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Systems

Carolyn R. Bartley, BS

2nd Lieutenant, USAF

March 2003

AFIT/GCS/ENG/03-03

A VISUAL LANGUAGE FOR COMPOSABLE SIMULATION SCENARIOS

Carolyn R. Bartley, BS
2nd Lieutenant, USAF

Approved:

Karl S. Mathias, Lieutenant Colonel, USAF Chairman	6 MAR 03 date
Timothy M. Jacobs, Lieutenant Colonel, USAF Member	6 MAR03 date
Raymond R Hill Raymond R. Hill, Lieutenant Colonel, USAF (Ret) Member	6Mar03 date

Acknowledgements

I would like to say thank you to everyone who helped me along the way. I would like to give special thanks to Lieutenant Colonel Mathias for helping me keep my direction and giving me the opportunity to conduct research in the field of composable simulations. I would also like to thank Lieutenant Colonel Jacobs for helping me discuss different aspects of the visual language developed.

Finally, I owe my family a big thank-you for their love and support. To my sister for still answering the phone when I called at 1 am, to my mom for listening to me when I was stressed out, and to my Dad for reminding me to "stay focused."

Carolyn R. Bartley

Table of Contents

Page

List of Figures

Figure Page

List of Tables

Abstract

Modeling and Simulation plays an important role in how the Air Force trains and fights. Scenarios are used in simulation to give users the ability to specify entities and behaviors that should be simulated by a model: however, building and understanding scenarios can be a difficult and time-consuming process. Furthermore, as composable simulations become more prominent, the need for a common descriptor for simulation scenarios has become evident.

In order to reduce the complexity of creating and understanding simulation scenarios, a visual language was created. The research on visual languages presented in this thesis examines methods of visually specifying the high-level behavior of entities in scenarios and how to represent the hierarchy of the entities in scenarios. Through a study of current behavior specification techniques and the properties of mission-level simulation scenarios, Simulation Behavior Specification Diagrams (SBSD) were developed to represent the behavior of entities in scenarios. Additionally, the information visualization technique of treemaps was adapted to represent the hierarchy of entities in scenarios.

After completing case studies on scenarios for the OneSAF simulation model, SBSDs and the application of treemaps to scenarios was considered successful. SBSD diagrams accurately represented the behavior of entities in the simulation scenarios and through software can be converted into code for use by simulation models. The treemap displayed the hierarchy of the entities along with information about the relative size of the entities when applied to simulation scenarios.

A VISUAL LANGUAGE FOR COMPOSABLE SIMULATION SCENARIOS

I. Introduction

1.1 Purpose

Over the next 20 years, the shift from "a threat-based force to a capabilities-based force" along with the "trend toward the information and cognitive warfare and away from platform-centric approaches" will cause modeling and simulation's role in the military forces to become increasingly important [DMS02]. However, the current state of simulation and modeling will not be able to fully support the future modeling and simulation needs of the military forces. As noted by a key study of the DoD transformation process, "A new generation of models and simulations will be needed to support distributed training; robust and continuous experimentation; operational planning, execution, and assessment tools [DMS02]."

One of the major difficulties with the current modeling and simulation process is the length of time it takes to complete simulation scenarios. A single scenario can take weeks or even months to build. Therefore, the purpose of this thesis is to develop a visual language for the representation of different aspects of simulation scenarios, which in turn will make simulation scenarios easier to compose and comprehend.

1.2 Background

Due to the large amount of simulation data already in existence and the large overhead in simulation and modeling, there have been several studies completed which focus on how to reuse simulation models (or components of the models) and how to get different simulations models to run together. A relatively recent trend in modeling and simulations is the study of "Composable Interoperability." The idea behind composable interoperability is two fold. Interoperability deals with the idea that multiple simulation applications with different levels of fidelity and communication protocols have the ability to work with each other. Composable simulations on the other hand, are simulations built out of predefined components (or entities) that can be modified and used by multiple simulations. The scenarios created for these simulations are defined by the components in the scenario and the relationship between the components. A component can be anything from the representation of a bolt on an engine, to the model of an army brigade. Although there are many benefits to composable simulations, there are also many challenges that have to be overcome before composable simulations become a reality. The lack of a common language describing the components and architectures of the components in simulation scenarios is one such obstacle.

Visual languages have been used in several other disciplines as a common descriptor for systems and the components that make up the systems. The aspects of systems and their components described by visual languages include structure, behavior, relationships, and communications between the components of the system. Visual languages have many benefits as diagrams can represent complex relationships and communicate structure better then text alone. They also aid in comprehension as they eliminate searching and support perceptual inferences [HOR98].

1.3 Research Focus

The focus of the research presented in this thesis is on how to best visually represent the scenarios created for simulations used by the Department of Defense. The visual representation of behaviors assigned to components and the hierarchy among the components in the scenarios are two aspects of scenarios that are addressed. Furthermore, the research works to identify other areas of composable simulations that can be better represented through visual representation.

1.3.1 Objectives

The overall objective of the research presented is to create a visual language that aids in the comprehension and composition of simulation scenarios. In particular, the language is intended to help in the development and comprehension of composable simulation scenarios at the mission-level by allowing for easier comprehension of the behavior of, and hierarchical relationships between, the entities in the scenarios. By modeling the components of simulation scenarios it is the hypothesis of this thesis that the visual language developed will increase the understanding of the structure and behavior of components used in simulation scenarios. Furthermore by having a common language to describe the simulation scenarios, the language serves as a basis for a tool that could potentially allow for one scenario to be generated for multiple simulation environments, or for the conversion of a simulation scenario from one environment into a simulation scenario for another environment.

The main objective has been broken into two sub-objectives. The first sub-objective is the development of a visual language that describes the assigned behavior of components acting as entities in simulation scenarios. By creating a visual language to describe the events that entities perform in a simulation, the visual language aids in the comprehension of already completed scenarios and the in the development of new scenarios.

The second sub-objective is to facilitate the navigation and understanding of hierarchies in mission-level simulation scenarios. This objective is achieved through the application of treemaps to the hierarchy of units and entities in mission-level simulation scenarios. By applying treemaps to the hierarchy of entities in the scenarios, users of the simulation can look at all the entities in the battlefield in a single space, without having to trace through long lists of units or maximize/minimize nodes of a tree. Furthermore, by customizing the properties of the treemap, such as color and the borders of the boxes used to represent the hierarchy, the user is given the ability to customize the treemap to present the desired information in a way that assists user comprehension.

1.3.2 Assumptions

This thesis focuses on representing the behavior and hierarchy of components in scenarios of mission-level models. The test simulation for the research is the OneSAF simulation used by the United States Army. By focusing the research to one level of simulation modeling it reduces the problems introduced by combining components of different levels of abstraction. However, the ideas presented in this research are applicable to all levels of simulation.

1.3.3 Approach

The approach used to conduct the research for this thesis consisted of four major steps. First, an in depth study was completed on the benefits, problems, and obstacles associated with composable simulation. Through the study of current simulation models and composable simulation models currently under development, the parts of assigned entity behavior in scenarios that need to be represented for comprehension and execution of the scenarios were identified. Concurrently, a review of methods used for behavior specification and process modeling in software engineering and simulation was conducted.

An evaluation of the benefits and limitations of the behavior specification methods when applied to the domain of behaviors assigned to entities in simulation scenarios served as a justification for the development of behavior specification diagrams presented in Chapter 4. The development of a visual behavior specification model for simulation scenarios was the second step of the research. During development, visual components were selected to represent the different aspects of the high-level behavior of the entities in simulations. Syntax and semantics were then added to the each of the components in order to present a complete visual language.

Third, to further justify why a new visual language was needed and to demonstrate that the language covers the different behavioral aspects of entities in simulation scenarios several case studies were implemented. The scenarios used for the case studies were selected to demonstrate how the current behavior specification models fail to handle different aspects of simulations, and how the new diagrams handle these aspects. Furthermore, the case studies cover several different areas of mission-level simulations to show that the diagrams developed cover a wide range of simulation behavior.

Finally, a program to support the creation and analysis of simulation scenarios was implemented. A GUI that allows users to assign behaviors to the entities in scenarios, through dropping and dragging components of the visual language onto the screen serves as the integral part of the program. The purpose of the program is to show that by using the behavior specification model presented in this thesis to visually represent different aspects of scenarios, composable simulation scenarios are easier to build and understand. The program implemented serves as a starting point for future research. A proposed final goal of the program is to have a tool that is able to generate code for one simulation or multiple simulations using the visual language as input.

In addition to the above steps the information visualization technique of treemaps was applied to the hierarchy of the entities in battlefield simulation scenarios. The application of the treemaps shows how different aspects of the scenario can be enhanced through modification of treemap properties.

1.4 Summary

In the world of simulation and modeling, composable simulations are the future. Currently, however, composable simulations are difficult to build and maintain due to their complexity and lack of standard representation. In the research conducted for this thesis, a visual language is applied to simulation and modeling in order to reduce the complexity of, and serve as a standard descriptor for, certain aspects of simulation scenarios. The validity and success of the language developed is shown through case studies and the implementation of a program that uses the visual language developed to graphically represent the behaviors of the entities in the simulation scenarios. The program also demonstrates the use of applying treemaps to the hierarchy of entities in simulation scenarios.

The next five chapters present the research and results of this thesis. Chapter 2, *Literature Review,* gives a background on composable simulations, visual languages, behavior specification techniques, and treemaps. Chapter 3, *Methodology,* presents the motivation, goals, evaluation techniques, and success criteria used in the research conducted for this thesis. Chapter 4, *Language Definition,* consists of a breakdown of the components of the visual language developed for behavior specification of entities and the application of treemaps to simulation scenarios. Case studies and the application of the visual langue to OneSAF scenarios through a Java program are discussed in Chapter 5, *Implementation and Case Studies*. The conclusion, along with ideas for future work is presented in Chapter 6, *Conclusions and Future Work.*

II. Literature Review

2.1 Introduction

The research of this thesis focuses on developing a visual language for composable simulations. In particular, the research deals with developing a visual language to describe the high-level behavior of entities in simulation scenarios and the application of treemaps to them in order to reduce their complexity. In order to form a basis for the language developed in Chapter 4 and to apply treemaps to simulation scenarios, several different research areas were examined. This chapter presents a review of these areas. The first area discussed is the domain of simulation and modeling. In this section a close look is taken at the benefits, problems, and challenges of composable simulations. The next research area presented are topics related to visual languages including the benefits of using visual languages, the components of visual languages definitions, and the Unified Modeling Language. Next, behavior specification techniques used in software engineering and simulation models are examined. Finally, this chapter concludes with a brief discussion on the information visualization technique of treemaps.

2.2 Modeling and Simulation – An Overview

As the problem that this thesis attempts to solve is in the field of modeling and simulation, it is important to gain at least a basic understanding of modeling and simulation. In the *Modeling and Simulation Master Plan* published by the Air Forces' Directorate of Modeling, Simulation, and Analysis, a model is defined as: "a physical, mathematical, or otherwise logical representation of a system, entity, phenomenon or process" [DMS95]. Mathematical models serve as an abstract representation of the system and provide a way of developing quantitative performance requirements. Static models are used to represent a state of a system, and dynamic models show conditions of the system that change with time. Models and simulations are used in several different domains including education and training, acquisition, operational planning, and experimentation [DMS95].

Models and simulations used by the United States military can be broken into a hierarchy based on the fidelity of the system and what is being modeled. At the bottom of the hierarchy are engineering models. Engineering models are used for design, cost, and manufacturing supportability. Engagement models sit one level above engineering models and assist in the evaluation of system effectiveness against enemy systems. At the next level, mission-level models measure the "effectiveness of a force package or multiple platforms performing a specific mission" [DMS95]. Finally, theater/campaign-level models predict the "outcomes of joint/combined forces in a theatre/campaign level conflict" [DMS95]. The research conducted for this thesis is on mission-level models, although the results are applicable to the other levels of models in the hierarchy.

Simulation plays an important role in military training and operation. Changes in technology will transform how the military forces organize, train, procure new weapons, and operate. In order to support this transformation, simulation must also change. A key study of the Department of Defense transformation processes noted that, "A new generation of models and simulations will be needed to support distributed training: robust and continuous experimentation; and operational planning, execution, and assessment tools" [DMS02]. One way that modeling and simulation is being transformed is by the development of composable and distributed simulations. In 1995, the Defense Modeling and Simulation Office (DMSO) initiated the DMSO High-level Architecture program, which served as the first step in addressing some of the issues surrounding the interoperability of simulation components which is an integral part of composable simulations.

2.3 High-level Architecture

The High-level Architecture (HLA) sponsored by DMSO provides a general software architecture for distributed systems. It was created under the idea that no simulation can satisfy all uses and users, and provides a structure to support the reuse of capabilities available in different simulations [DAH98].

The reusability of HLA is based around the concept of an HLA federation, which is “a composable set of interacting simulations,” and allows for simulations “developed for one purpose to be applied to another application” [DAH98]. A federation consists of the three functional components depicted in Figure 1. The simulations, referred to as federates, make up the first functional component of a federation. All object representation is done through federates and each federate must contain specified capabilities that allow the objects in one simulation to communicate with objects in another simulation through the runtime infrastructure (RTI). The RTI is the second component and serves as a distributed operating system for the federation. The RTI supports the simulations by providing a set of services that carry out federate-to-federate interactions and federation management support functions. The runtime

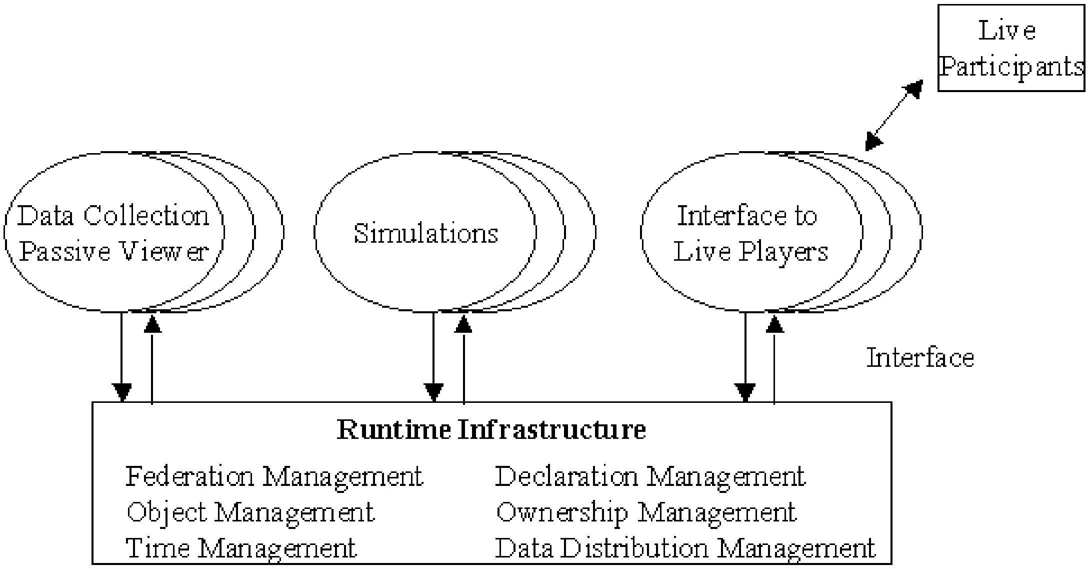


Figure 1. Components of the High-level Architecture [DAH98]

interface makes up the third component of the HLA federation. The runtime interface provides standard methods for federates to interact with the RTI, to invoke the RTI services, and to respond to requests from the RTI. The HLA also supports the passive collection of simulation data and monitoring of simulation activities and interfaces to live participants [DAH98].

HLA is formally defined by the interface specification, the object model template (OMT), and the HLA rules. “The HLA interface specification describes the runtime services provided by the federates to the RTI” while, “HLA object models are descriptions of the essential sharable elements in the simulation or federation in ‘object’ terms” [DAH98]. There are no constraints on the content of an object model. However, each federate and federation must document its model using a standard object model specification. The specification provides open information sharing across the simulation community to facilitate the reuse of simulations [DAH98].

Despite the work that HLA has done to promote reusability and composability among simulation models it is commonly agreed that the architecture provided by the HLA is not strong enough to provide true composability in simulations. Furthermore, HLA is difficult to understand and has a steep learning curve. Finally, HLA is used to describe the architecture of the simulation and the components used by the simulation, not the scenarios created for the simulations.

2.4 Composable Simulations

Beyond HLA lies the idea of composable simulations. The concept of composable simulations is one of the current focuses of not only DMSO, but the field of modeling and simulation as a whole. The following sections provide a brief overview of what composable simulation is, the benefits that can be achieved from composable simulations, the drawbacks of composable simulations, and some of the challenges that need to be met before composable simulations can become a reality.

2.4.1 Definition of a Component

In *Proposal For Composable Modeling and Simulation Studies* produced by DMSO the following definition of components is given:

> Components are similar to classes, but generally their implementation is completely hidden. Components may be implemented by a single class, more than one class, or even by a traditional procedure or function in a non-object oriented programming language. Components also conform to the standards defined by a component model [DMS02]

Components were also defined at the Composable Modeling and Simulation Workshop. At the workshop a group discussing the concept of operations for composable simulations listed the following properties of a components [DMS02b]:

1. May be used by other software elements (clients).
2. May be used by clients without the intervention of component developers.
3. Includes a specification of all dependencies (hardware and software platform, versions, other components).
4. Includes a precise specification of the functionalities it offers.
5. Is usable on the sole basis of that specification.
6. Is easily composable with other components.
7. Can be integrated.

The group also noted that:

1. Components are NOT objects in the OO sense.
2. Not just software (data too), not just module level.
3. Open source desirable but not required.

Other articles discussing composable simulation also gave more definitions and properties of components [DMS02d], [DMS02e], [BID00]. However, despite the differences between the definitions of components, they all share similar themes. First, components are entities that can stand on their own. Second, components provide an interface by which they interact with other components and parts of the system. Therefore, based on these properties two components that have the same interface should be able to be interchanged in a system with minimal impact on the implementation of the system. The third common property is that complex components are built from simpler components which allows for components to act as building blocks.

2.4.2 Benefits of Composable Simulations

According to Kasputis and Ng in their research on composable simulations presented in *Model Composability: Formulating a Research Thrust: Composable Simulation*s [KAS00], composable simulations have the potential to offer several benefits to the simulation community. These benefits include providing higher-quality simulations, lowering the development time of simulations, and lowering the cost of simulations [KAS00].

Kasputis and Ng indicate that there are five main ways composable simulations can contribute to higher quality simulations. First, composable simulations will provide a higher comprehensiveness of simulations. In simulations, comprehensiveness is the ability of a simulation to "address all aspects of the problem under investigation and represent all potentially important factors within the mission space" [KAS00]. Due to time, budget, and constraints of knowledge of real world systems, simulations often lack the comprehensiveness that the user would like or need. By promoting re-usability of algorithms, information, and models through the use of components, redundancies between the models can be reduced and more time can be spent on developing other desired aspects of the real world [KAS00]. The programming language Java supports the reusability of algorithms and models by providing standard libraries that support commonly used elements such as lists, strings, and hash tables. Developers using Java save time by tailoring the Java elements to their need, instead of re-creating the element.

Next, composable simulations will help to provide consistency and improve the validity of simulation models. Consistency is needed as simulations are becoming multi-resolution, because the outcome of the simulations must not change as the models are run at different levels of fidelity. Composable simulations can support consistency by providing detailed module descriptors and by allowing for complete and proper testing of the library of modules. Improved validity of the simulations would produce higher quality simulations as validation and verification efforts could be concentrated on the library of components for a system [KAS00].

Two other major benefits that composable simulations have to offer are a quicker production time and a lower cost for simulations. One key factor in decreasing the time it takes to build simulations is reducing the setup and initialization time of simulations scenarios. In [KAS00] Kasputis and Ng indicate that “many current simulations take considerable time and effort to setup and initialize.” Through composable simulation setup time can be reduced as users can reuse pieces of code, making modifications as needed. Second, the time for analysis of simulation results can be reduced as composable simulation are envisioned to have the capability to produce only the desired information, by filtering out the data the user does not want and calculating user selected statistics. The lower costs that composable simulations have the potential to provide come from the reuse of software and software designs and potentially lower maintenance costs [KAS00].

2.4.3 Drawbacks to Composable Simulations:

Despite the benefits of composable simulations, composable simulations are difficult to develop. When looking at the price of composable simulations, members attending the Composable Modeling and Simulation Workshop identified several factors contributing to the cost of composable simulations. First, they indicated that it is more difficult and expensive to build simulations that are composable. They also questioned if the configuration management cost of composable simulations would be lower. Furthermore, although the life cycle of the composable simulations is shorter, it may not be short enough to compensate for the higher initial cost and the higher cost of general purpose applications. Finally, the members noted that working with composable simulations required the developer to understand networks and other things outside their component [DMS02b].

2.4.4 Barriers to Composable Simulation

In addition to the costs of composable simulations there are many barriers that need to be crossed before composable simulations can become a reality. In the article, “A Model-Based Approach to Simulation Composition,” Aronson and Bose identify four major sub-problems related to simulation composition [ARO99]:

1. Capturing a simulation scenario defining the target system
2. Constructing a software-based structural model
3. Selecting components which satisfy the structural model and global non-functional constraints
4. Determining the inter-component coordination

The first sub-problem deals with conversion of requirements to relevant domain entities, activities, and behaviors used in the simulation [ARO99]. This problem is similar to the problem of going from use cases to system design, as found in software engineering.

The second sub-problem of “constructing a software based structural model” is concerned with the problem of matching the functional elements in the simulation scenario to a structure where all the components are connected correctly, interoperate consistently, and still stratify the functional requirements [ARO99]. The problem exists because it is possible to have components that individually meet the functional requirements, while the combination of the components does not.

The third sub-problem deals with how the user selects the best component if a large repository or several distributed repositories are available. With a large number of available components the correct component needs to be selected easily. This sub-problem parallels searching the Internet, in that there is a lot of information available on the Internet, but it is not always easy to find the information that best fits the need of the user.

The final sub-problem addresses the issue of how to deal with the overall synchronization of the components and the communication between the different components in composable simulations [ARO99]. This problem focuses on the architecture of composable simulations and the communication that occurs between the components in the architecture.

Another challenge faced by composable simulations is the lack of guiding principles. Kasputis and Ng believe that “unless models are designed to work together – they don’t” [KAS00]. In the past, simulations were built for one simulation program without regard to how they might work with another simulation. The set of simulations used by the Department of Defense is an example of this occurrence. Currently there are multiple mission-level simulations, but because of the way each system was built, it is very difficult to make the simulations work together. HLA is an attempt to get the simulations to work together, but has been shown to be less than an ideal solution.

The research presented above shows that although developing composable simulations is a difficult engineering problem, the potential benefits are quite good. However, in order for composable simulations to be beneficial many challenges have to be met. Many of the problems and challenges composable simulations face parallel the problems and challenges found in software engineering. Software engineering deals with classes, objects, object behavior, components, system behavior, etc. In order to help reduce the complexity brought on by the size and domain of the systems being built the software engineering community has developed visual languages to help developers better understand software systems, the pieces that compose systems, and the behavior of systems and components within the systems. Therefore, a visual language, becomes an ideal candidate to not only help reduce the complexity of simulation scenarios, but to also help provide unity and a common guideline on which to build components used in composable simulation scenarios.

2.5 Reasons to Use Visual Languages/Diagrams

Every day people use diagrams to communicate information. Diagrams have several important functions and benefits that make them the ideal choice to communicate certain types of information. First, diagrams represent complex relationships better then text alone [HOR98]. Problems that occur when the number of novel elements and their connections get bigger than the capacity of short term memory can be solved by using diagrams. Second, as diagrams help to “organize and manage problems and issues” of abstract elements, they make the abstract concrete [HOR98]. Diagrams also easily show concepts, like changes in time and branching that are difficult to communicate through written prose. Furthermore, structure is often communicated more efficiently with a diagram, than through text alone. For example, a family tree is simpler to understand and communicates the same relationships much more effectively than a text paragraph relating the same information [HOR98].

For several reasons, diagrams are better than prose for many types of information. First, because diagrams “group together all the information that is used together” searching for elements needed to solve a problem is eliminated [HOR98]. Second the matching of symbolic labels is avoided as diagrams group information about single elements. Diagrams also “support a large number of perceptual inferences” and “help the learner build runnable mental models” that portray “each major state that each component can be in and the relations between a state change in one component and the state changes in other components”[HOR98]. Finally, diagrams are better representations of knowledge because they are computational in nature and, “the indexing of information supports useful and efficient computation processes” [HOR98].

When applied to the representation of behavior in simulations there are also several reasons to use diagrams. The developers of the real-time object-oriented modeling (ROOM) method believe that “graphics based representations facilitate communication among all parties involved in system development” [SEL94]. For example, state machines are traditionally represented as graphs since insight is provided faster through the graphical representation then the corresponding textual or tabular representation [SEL94].

However, diagrams are not perfect. Every type of diagram has its own syntax and semantics which the user must learn before they can understand it. The more complicated the syntax and semantics the bigger the learning curve is, and the larger the room for errors. Furthermore, often several diagrams may be needed to accurately represent all aspects of what is being modeled [HOR98]. The Unified Modeling language is one example of this, as several diagrams are used to model the multiple aspects of software systems. When several diagrams are used to model the same system, problems with consistency among the diagrams and the relationship between the diagrams can occur. Finally, graphical representations are impractical for capturing detail as graphics that are overloaded with detail can become as difficult to understand as the corresponding text [SEL94].

When developing graphical modeling tools one should also be aware of the problems and limitations that graphical modeling can inflict on the model and modeler. First, graphical modeling tools can force a model to "fit within a rigid framework bounded by available icons, menus, and forms" [CRA98]. The rigid framework can limit the versatility of the model, making it inaccurate. Furthermore, the more screens of icons and links a model is composed of the more cumbersome it becomes to view, edit, and document the model because the user must navigate through mazes of icons, menus, click-buttons, data fields, and code segments [CRA98].

2.6 Syntax and Semantics of Languages

Every well-defined language, whether a visual or textual language has its own syntax and semantics. Syntax and semantics are the components that define a language. The UML 1.4 specification states that, "The syntax defines what constructs exist in the language and how the constructs are built up in terms of other constructs " [OMG01].

The notation independent definition of the syntax is the abstract syntax of the language, while the concrete syntax is defined by mapping the notation onto the abstract syntax [OMG01]. Semantics, which gives meaning to the constructs, can be broken up into static and dynamic semantics. Static semantics describe how instances of constructs should be connected together to be meaningful, while dynamic semantics determine the meaning of a well-formed construct. The semantics of a language is given by a mapping from the syntax domain, to a semantics domain [CLA99].

2.7 Unified Modeling Language

The Unified Modeling Language (UML) is one example of a visual modeling language. Over the past decade the Unified Modeling Language has become the standard modeling language for modeling software systems. UML allows users, through twelve different types of diagrams, to model the static application structure of a system, the different aspects of dynamic behavior of a system, and the structure and organization of the application models [OMG02]. The different types of diagrams can be broken down into the following categories [OMG02]:

Structural Diagrams: Class Diagram, Object Diagram, Component Diagram, and Deployment Diagram.

Behavior Diagrams: Use Case Diagram, Sequence Diagram, Activity graph, Collaboration Diagram, and Statechart Diagram.

Model Management Diagrams: Packages, Subsystems, and Models

The diagrams are based on the concept of an object. An object is a “thing” that can be interacted with. The state of an object represents all the data an object stores and is represented through the attributes of an object. An object also has behavior which is the way an object acts and reacts in terms of state changing and message passing. Finally every object has an identity, which identifies an object independent of the values of the object's attributes.

In UML, objects belong to classes, which describe a set of objects "with an equivalent role or roles in a system" [OMG2]. Objects belonging to the same class share the same set of attributes (the values of the attributes can be different) and behaviors. Classes are related to each other by using associations. Through associations multiplicity (how many objects of class A is class B related to), aggregation (a contains relationship), generalization (class B inherits all the properties of class A) and specialization can be shown [OMG02].

UML is not limited to software systems, but can also be applied to many other applications including real-time systems and business applications. With increased use of UML and the application of UML to areas outside of software systems, UML faces several challenges. Some of the major challenges revolve around the size and complexity of UML, while other challenges focus on the limitations and ambiguity of UML.

2.7.1 UML Extensions

An important part of the specification of UML is the understanding that it cannot possibly meet every user's need. Therefore, the specification of UML includes extension mechanisms that allow UML to be adapted to meet specific needs of a domain. As the visual language being developed for composable simulation scenarios is designed to be applied to multiple simulations it faces many of the same problems as UML regarding complexity and ambiguity. Therefore, the ability of the language to be extended should be looked at, and the extensions that UML defines serve as an excellent basis for defining the extensions.

The Extension Mechanisms package in UML allows for UML model elements to be customized or extended. Extensions can be grouped together for a specific purpose to form a profile. Through the Extension Mechanisms package users can extend UML by creating new kinds of modeling elements and attaching free-form information to defined modeling elements.

The Extension Mechanisms package is made up of profiles, stereotypes, tag definitions, and constraints. The UML 1.4 specification defines profiles as “a stereotyped package that contains model elements that have been customized for a specific domain or purpose by extending the metamodel using stereotypes, tagged definitions, and constraints” [OMG01]. Profiles may also specify the model libraries it is dependent on and the metamodel subset it extends. Typical profiles consist of a list of new stereotypes, with a definition for the stereotype and graphical notation (if graphical notation has been added) [FLO02].

A stereotype is defined as “a model element that defines additional values, additional constraints, and optionally a new graphical representation” [OMG02]. When a stereotype is applied to a model element, the values and constraints of the stereotype are added to the attributes, associations, and superclasses that the element has in the standard UML. A definition for a stereotype may be informal, described via text, more formal

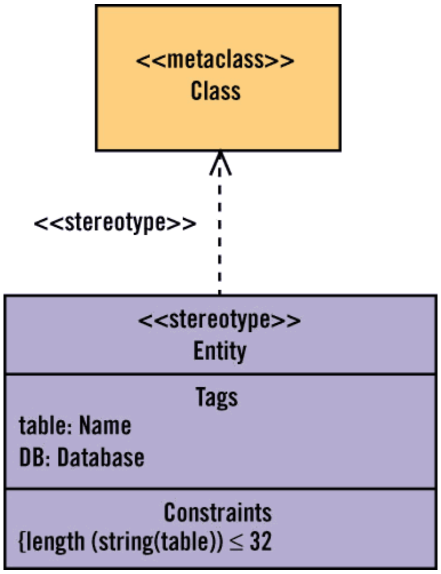


Figure 2. UML Stereotype [FLO02]

with a UML graphic, or have tabular notation. Figure 2 is an example of a stereotype for a database entity, defined through a UML graphic.

Stereotypes can be represented by the default practice of keywords or have a graphical notation. However, graphical notations should be added sparingly, because it is difficult to design new notation that is both consistent with the old notation and mnemonic [FLO02].

Tag definitions are used to “specify new kinds of properties that may be attached to model elements” [OMG01]. Tagged values, which may be simple data types or a reference to other model elements, are used to specify the actual properties of individual model elements. Constraints are used to refine semantics of model elements. When constraints are applied to a stereotype, all elements branded by the stereotype must obey the constraints.

Profiles offer several benefits to users of UML. First, they allow groups of people to define extensions for their purposes without having to go through the UML standards process. Second, because the profiles do not need to go through the UML standards process they are not added into the already large and complex specification [FLO02]. Profiles also refine UML for specific domains, reducing the complexity of UML for users and allowing for complete semantics to be defined for specific domains.

Another way that UML can be extended is to create new metaclasses and other meta constructs. The OMG Meta Object Facility (MOF) allows in principle any metamodel to be defined. By using tools and repositories that support the MOF, users can create new meta models based on UML. These models differ from profiles in that the restrictions placed on UML profiles to ensure that the profiles are "purely additive" do not apply to metamodels [OMG01].

2.7.2 UML and Simulation

The application of UML to simulation and simulation scenarios is not new and several studies on how UML can be applied to composable simulations have been conducted. One study conducted by Richte and Lothar proposes that through UML "a description of the requirements and performance features of the simulation model regarding its structure and dynamics" can be obtained by using different types of UML diagrams [RIC00]. The study indicates that there are several benefits to applying UML and the unified process to simulations. These benefits include [RIC00]:

1. Establishing a general standard for modeling and documentation of simulation models.

2. Independence from the used simulation software, i.e., the way the model is encoded.

3. Definition of a methodology for developing simulation models through the use of object-oriented modeling and documentation.

4. Visualization of concepts, structures and dynamics of a simulation model through UML.

5. Identification of reusable components in simulation models.

6. Building a framework for project management for simulation studies.

Another study by Stytz and Banks applies UML to the building of HLA simulations [STY01]. In the study UML was applied to simulations in order to reduce the difficulty of development, use, and re-use of simulation models. By using UML to visually represent the federation and federates, Stytz and Banks believe users will be able to comprehend the operations, objects, and parameters used in the simulation without having to be fluent in HLA. They also believe that the improved documentation will help with the validation of the federation. Stytz and Banks concluded that by having a standardized set of UML documentation the following objectives can be achieved:

1. Enable better management of the federation simulation environment process and improve the description of all a federate's or federation's capabilities and requirements. Improve the capability to exploit advances made in simulation-related technologies

2. Help describe the system to non-technical users

3. Give the simulation improved capability to document a federation's functionality

The methodology described in [STY01] approaches the use of UML in the HLA architecture by using UML to design the federation simulation environment by documenting with UML all the behaviors and properties that the environment must possess. Then, from the UML documentation, the four Federation Object Model (FOM) tables which are required for HLA simulations are created and used in an iterative process to refine the UML based design. A FOM in HLA "describes the set of objects, attributes, and interactions which are shared across a federation" [DAH98].

The process of generating the FOM tables from UML diagrams consists of three main steps: the determination of the use cases, the determination of the static model of the simulation, and the determination of the dynamic behavior of the system. How the federates and federations are used are documented through use cases, while class diagrams and object diagrams document the major components in the simulation, their static attributes, and relationship between each other. The behavior, or dynamic modeling, of the system is done through sequence and collaboration diagrams [STY01]. Figure 3 shows the order of the diagrams and how they are related to the four FOM tables.

Stytz and Banks also concluded that several other areas of FOM development can be improved or helped by the application of UML in the above manner. These areas include:

1. The documentation of the expected behaviors and characteristics of computer-generated actors
2. The required performance of the network that supports the federation
3. The development of documentation to address network latency, bandwidth and the real-time requirements the data must be transmitted across the network
4. Determining the virtual circuits that should be established to obtain the desired quality of service
5. Documentation of the required levels of detail
6. Standardization of common notations

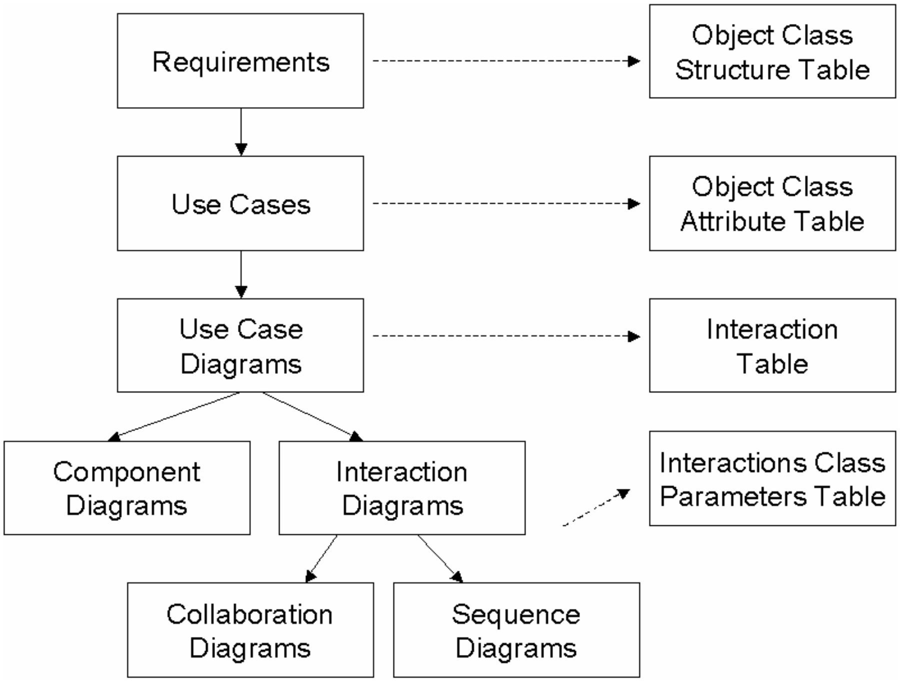


Figure 3. Mapping of UML Diagrams to FOM Tables [STY01]

Despite these benefits, UML currently is not completely suited to model simulations or simulation scenarios. One suggestions made for future research in [STY01] is that research should be done to develop "HLA specific extensions to UML that will support a more formalized and standardized description of the timing-related performance, accuracy, dead-reckoning, and other open federation and federate design and documentation issues."

2.7.3 Differences Between Software and Simulations

Although software systems and simulation scenarios share many of the same similarities there are several key differences. First, in the design of software systems, the focus is on the relationship between the classes in the system, not on actual instances of classes. In the UML there are several diagrams designed to show the relationship between different classes, yet very few diagrams focusing on actual instances of the classes in the system. However in a scenario, the focus is on the relationship, value, and behavior of the entities in the simulation, which are much more similar to objects (instances of a class) then to classes. For example in a scenario there may only be three types of entities (types of entities would be equivalent to a class), yet there may be a hundred instances of each type of entity in the scenario. Furthermore, when building a scenario the value and behavior of each of the entities in the scenario is important.

Second, in software systems, all instances of a class contain the same behavior definition. It is the attributes of the instance combined with the behavior definition that determine how the instance reacts to a function call. However, in certain simulation scenarios, the builder of a scenario can assign different instances of the same type of entity different sequences of behaviors, in the same simulation.

Third, the behavior of classes described in UML is different from the behavior of entities defined by the user in simulation scenarios. In software engineering the behavior specification of an object is focused on the different states an object can be in and how a state transitions from one state to the next. In simulation scenarios the behavior assigned to an entity by the user is a sequence of activities, the conditions between the activities, and the parameters of the activity. Although the types of entities in simulation scenarios have defined behavior similar to classes, the behavior is complex and hard-coded by software engineers and subject matter experts. The focus of the scenario is to define what activities the entity performs and the sequence they are performed in, not so much what activities they can perform.

Finally, when talking about scenarios of mission-level simulation the active elements in the scenario follow a standard hierarchy. Typically every entity is in a chain of command. In software engineering there are multiple different types of system architectures a system can have.

One can also apply a hierarchy to components. Although this hierarchy can be applied to some software systems, a hierarchical architecture is not a general property of software. The hierarchy exists in that a complex component can be seen as a parent to the components that are put together to make the complex component. Each of these components can potentially be parents to other components. For example, component A might be composed of component B and component C. Component C is then composed of component D and component E. Component D is a simple component, but component E is composed of components F and G.

2.7.4 Why UML is not Ideal for Composable Simulations

At the Composable Modeling and Simulation Workshop it was identified that a language was needed to serve as an architectural-level description language for composable simulations. One of the languages considered was UML. A group examining the issue found that:

> UML is the defacto standard for capturing software requirements and static design. UML is well integrated, mature on visual presentation. UML has important ambiguities in the relationships in the differing views. These ambiguities are worked around by manual transformation and informal communications in current use. The composable components approach can founder on these ambiguities [DMS02c].

In light of the above statement the group went on to evaluate UML as the standard of choice for describing the architecture of composable simulations. The consensus of the group was the UML in its current state was not a complete solution for the following reasons [DMS02c]:

1. UML is the de facto standard for software design. However, languages are required at several levels: programming, design, architecture, and semantic meaning. UML is not a complete solution to the entire problem.

2. UML is mature in its visual representations, but is very immature and poorly defined in terms of the semantics of relationships between views (class, interaction, sequence, etc.). Question to be asked: What is the semantic relationship between the diagrams?

3. Works well as a standard for communicating designs among a group. Does not work well at abstraction levels above design.

4. Does the monopoly of UML inhibit other tools/languages from gaining acceptance?

5. UML has numerous limitations/ambiguities. These limitations may not affect many applications developed under it, but component-based simulation tends to magnify these deficiencies.

6. Other issues:

    a. Poor ability to aggregate low-level software designs into components, or to aggregate components into higher-level components.

    b. UML tends to focus on objects, but components are typically composed of numerous objects, or may have been developed using structured programming approaches.

    c. Temporal description aspects of UML are lacking

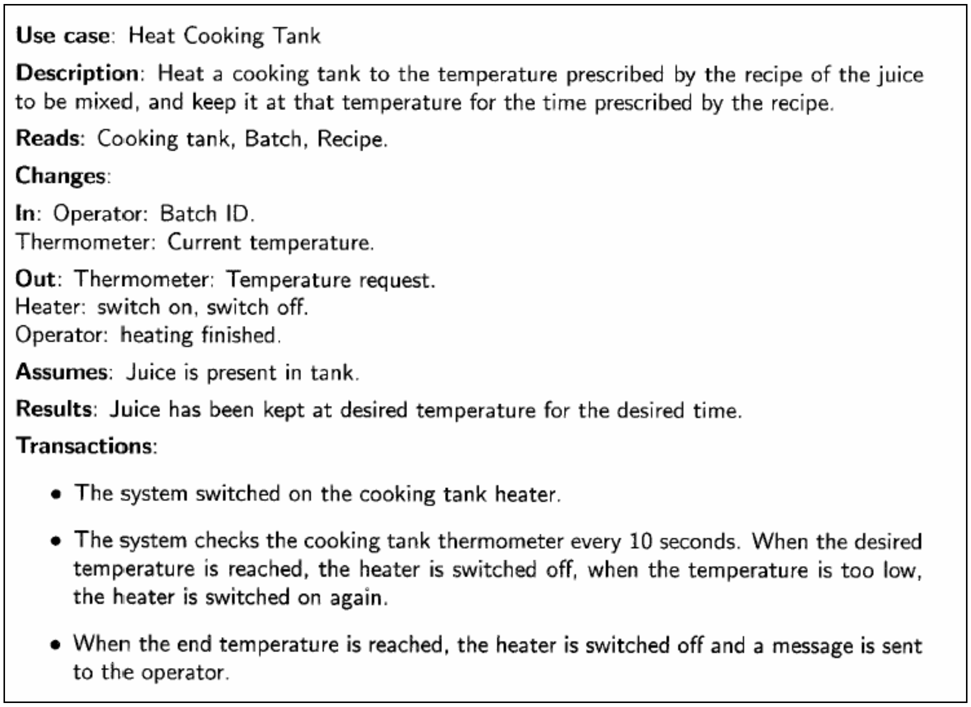
2.8 Representing Behavior

Representing behaviors is not a new problem, and there are several different behavior specification techniques. The following section gives an overview of some of the models currently being used, or models that have been applied to software engineering and simulations in the past.

2.8.1 Object-Oriented Software Behavior Specification Methods and Techniques

Behavior specification techniques are those that “can be used to show how functions of a system or of its components are ordered in time” [WIE98]. Several different graphical models have been used in software engineering and simulation to specify the behavior of the objects/entities in systems. These models include: process graphs, Jackson Structured Programming process structure diagrams, finite state transition diagrams, extended finite state transition diagrams, Mealy machines, state charts, process dependency diagrams, ROOMCharts, activity graphs, activity cycle diagrams, Petri nets, hierarchical finite state machines, and logic diagrams.

In these diagrams, states are represented by labeled nodes and transitions by directed edges. The labels on the edges indicate the events that trigger the transition, the actions generated by output, and potentially the actions performed on local variables.



Use case: Heat Cooking Tank

Description: Heat a cooking tank to the temperature prescribed by the recipe of the juice to be mixed, and keep it at that temperature for the time prescribed by the recipe.

Reads: Cooking tank, Batch, Recipe.

Changes:

In: Operator: Batch ID.
Thermometer: Current temperature.

Out: Thermometer: Temperature request.
Heater: switch on, switch off.
Operator: heating finished.

Assumes: Juice is present in tank.

Results: Juice has been kept at desired temperature for the desired time.

Transactions:

- The system switched on the cooking tank heater.
- The system checks the cooking tank thermometer every 10 seconds. When the desired temperature is reached, the heater is switched off, when the temperature is too low, the heater is switched on again.
- When the end temperature is reached, the heater is switched off and a message is sent to the operator.

Figure 4. Schema for the Heat Cooking Tank Use Case [WIE98].

Figure 4 gives the schema for the Heat Cooking Tank use case that will be represented by each of the discussed models. The schema and diagrams used to discuss the diagrams are published in [WIE98].

2.8.2 Process Graphs

Process graphs are directed graphs with labeled edges. The nodes of the process graph represent states while the edges represent state transitions. There may be infinitely many nodes, and from each node infinitely many edges in a process graph. Process algebra and dynamic logic use process graphs as interpretation structures for formal specification. Figure 5 gives an example of a process graph for the schema presented in Figure 4.

2.8.3 Process Structure Diagrams

Process Structure diagrams (PSD) use a tree diagram to visually represent a regular expression. Leaf nodes in the tree represent atomic actions, while interior nodes represent non-atomic processes. Sequences are represented by a left to right ordering of the nodes. Nodes labeled with a ○ indicate an alternative, while an * in the box represents iteration. Jackson Structured Programming uses PSDs to represent the

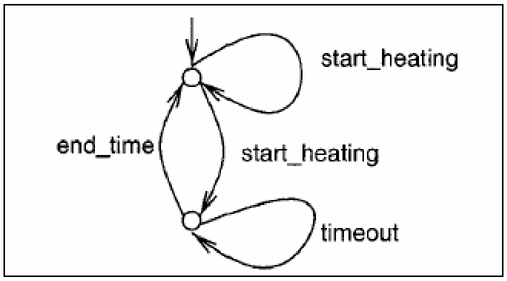


Figure 5. A Process Graph [WIE98]

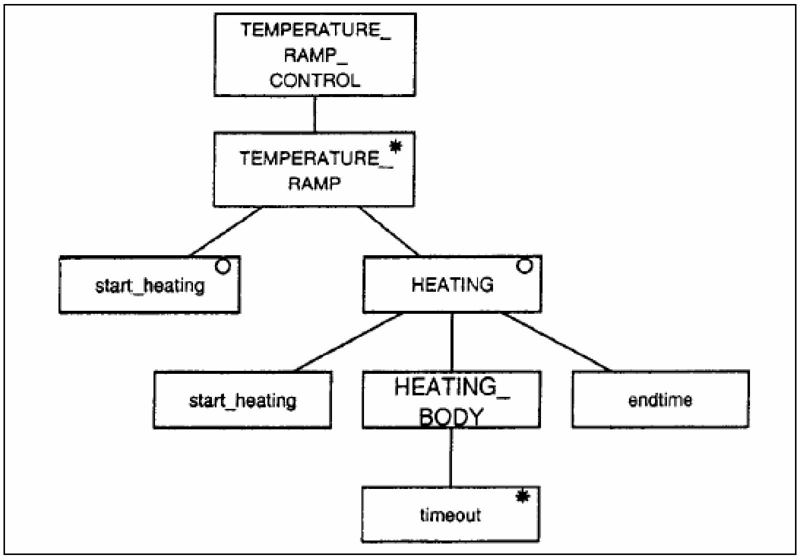


Figure 6. A Process Structure Diagram [WIE98].

lifecycle of entities and specify functions of the system. Figure 6 gives an example of the PSD for the schema in Figure 4.

2.8.4 Finite State Transition Diagrams

Finite State Transition Diagrams (FSTD) are directed graphs with a finite number of labeled nodes and edges. As in a process graph, the edges represent transitions while the nodes represent states. Ideally, because a FSTD contains a finite number of edges

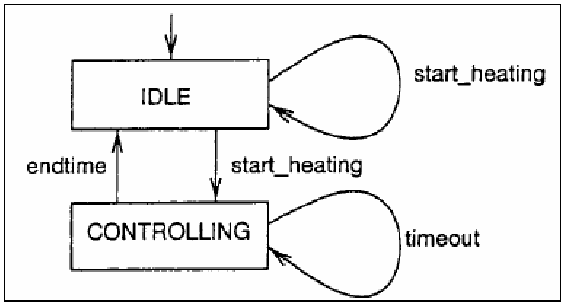


Figure 7. A Finite State Ttransition Diagram with Labeled States [WIE98]

and nodes that can be drawn on a finite piece of paper, as opposed to regular state transition diagrams which can have an infinite number of nodes and edges. Figure 7 gives an example of a FSTD for Figure 5.

2.8.5 Extended Finite State Transition Diagrams

Extended finite state transition diagrams increase the number of states the system can be in by containing variables that can be tested and updated by the finite state machine. The state of the system (the *global state*) is represented by the state of the nodes (the *explicit state*) combined with the value of the variables (the *extended state*). The variables can be local or external variables. Local variables are declared with the specification of the FSTD. The scope of the variables can be the entire STD, a transition, or a state. External variables are declared outside of the STD and are accessed through special operations. With the addition of variables the specification of state changes can be refined in the following ways [WIE98]:

1. The value of the variable might be changed by a state transition
2. A guard may be added to each transition specifying when the transition can occur.
3. Tests to determine what the next state will be can be added to the state machine. These tests can be used to eliminate non-determinism.

2.8.6 Mealy Machines/Mealy STD

State machines interact with their environment through input events (communications from the environment to the state machine) and output actions (communications from the state machine to the environment). Inputs events are associated with a transition and trigger the associated transition provided that a transition guard does not block the transition.

Output actions in Mealy machines are associated with transitions. The input events and output actions of a transition are separated by a horizontal line, while guards are placed in brackets following the events. Figure 8 is an example of a Mealy STD. Mealy machines also contain decision states, which are states where data from the external system is requested and then used to determine the next state of the system.

2.8.7 Moore Machines

Moore Machines are similar to Mealy machines except that outputs are associated with states instead of transitions. Therefore, actions are performed upon entry of the state and generally all transitions entering a state will generate the same output. The Shaer-Mellor method uses Moore machines for behavior specification. Figure 9 is an example of a diagram of a Moore machine.

2.8.8 State Charts

State charts are hi-graphs with Cartesian products, but no intersections. The nodes represent states and the directed hyper edges represent the state transitions. A state

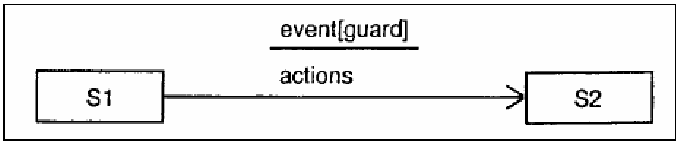


Figure 8. STD of a Mealy Machine [WIE98].

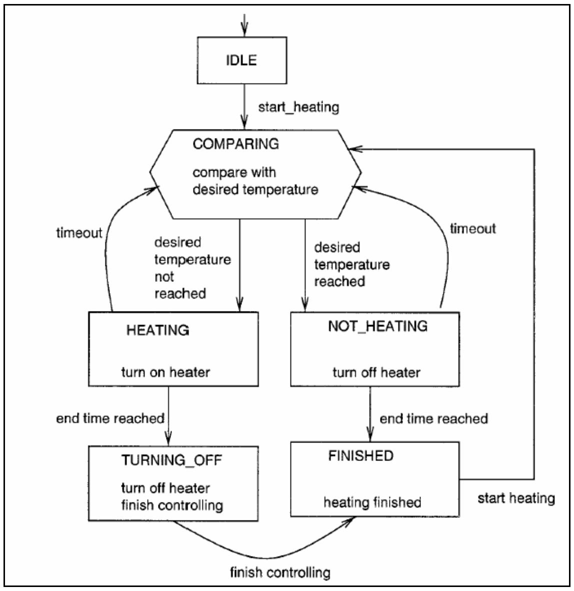


Figure 9. STD of a Moore Machine with a Decision State [WIE98]

can be partitioned into sub states through node inclusion, and parallelism is allowed through Cartesian products. Furthermore, state charts use local variables to represent an extended state and allow for actions to be specified along transitions, upon entry into states, and upon the exit from states. In the case that parallel composition is used, an action generated by a state transition is broadcast to all of the parallel components in the same state chart. Some models, such as Statemate have defined special events, actions, and conditions that can be used in state charts. Formal semantics of state charts have been defined by Herl et. al. and Pnueli and Shalev. Figure 10 shows a state chart of the process defined in Figure 4.

2.8.9 Process Dependency Diagrams

Process dependency diagrams are directed hypergraphs with nodes representing ongoing activities (processes) and the edges representing the transitions between the activities. Information engineering uses process dependency diagrams to represent precedence

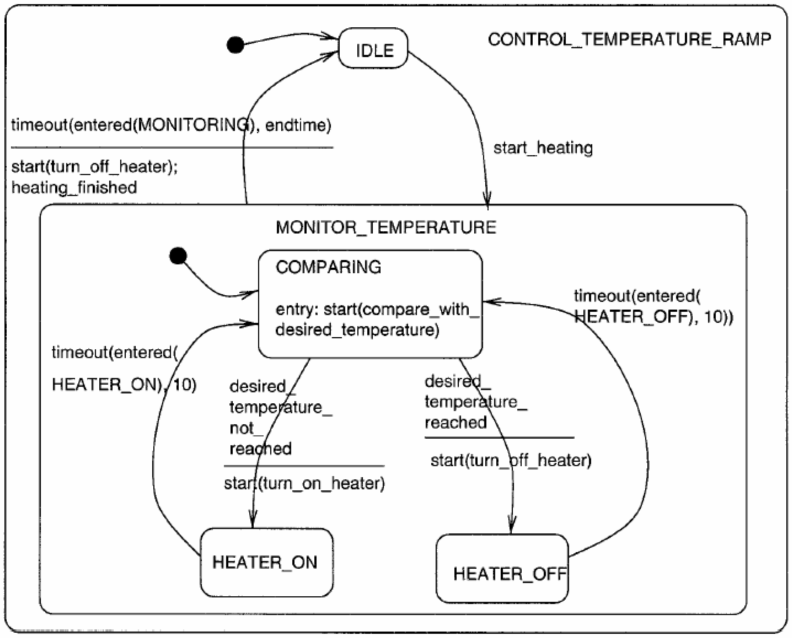


Figure 10. A State Chart [WIE98]

relations between activities. Process dependency diagrams contain conventions for parallel execution, alternative execution of processes, and cardinality properties of precedence relationships. They can also represent triggering events. Process dependency diagrams are similar to dataflow diagrams, except in a process dependency diagram the transitions represent a logical precedence and not a dataflow. Figure 11 gives an example of a process dependency diagram.

The Martin-Odell method and UML both use conventions related to the conventions in process dependency diagrams. The Martin-Odell method uses event diagrams, which are like process dependency diagrams without explicit representation of cardinality and event arrows, for behavior specification. The exact nature of the dependencies is depicted by control conditions. UML contains activity graphs which are similar to process dependency diagrams without events or cardinalities. They are defined

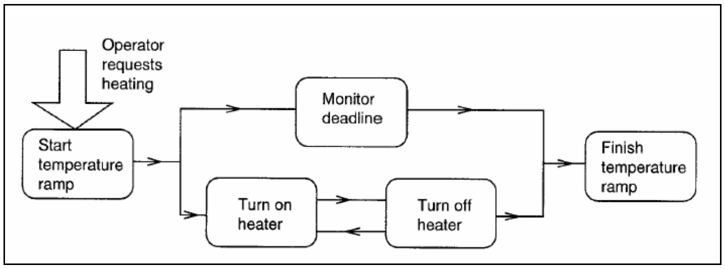


Figure 11. A Simple Process Dependency Diagram [WIE98].

as state charts where the states represent activities and the transitions are triggered when an activity is terminated.

2.8.10 ROOMCharts

The real-time object-oriented modeling (ROOM) method is used to model real-time systems. The idea behind ROOM is to break real time systems into a "hierarchical collection of components called actors" [SEL94]. An actor in a ROOM model is "a logical component of a system that can be active concurrently to the other actors in the system" [SEL94]. In ROOM actors must have a defined purpose, and an actor can be a software object with its own thread of control or a physical component that can behave independently of other objects. The behavior of actors in the system is represented by ROOMCharts.

A ROOMChart is a graphical representation of an extended state machine based on the state formalism of David Harel. The diagram consists of states represented by rounded rectangles and transitions represented by arrows. A state in a ROOMChart represents "a period of time during which an actor is exhibiting a particular kind of behavior" [SEL94]. The transitions in a ROOMChart are triggered by the arrival of messages (through the defined interface of the actor). Every transition must have a trigger and the trigger may contain an optional guard function. The guard function must evaluate to true or false, and the guard function must be true in order for the transition to be taken. Transitions can be given a label and be defined separately from the ROOMChart. The activities performed by the actor are defined by attaching actions to states or transactions. Actions, or tasks for the actor to perform, can be attached to transactions or to states as entry or exit actions. These actions are indicated in the form of statements written in executable instructions. Figure 12 is an example of a ROOMChart.

2.8.11 UML Activity Graphs

In UML, state machines are used to "specify behavior of various elements that are being modeled," and provide a foundation for activity graphs [OMG01]. An activity graph is an extension of a state machine and is used to "model processes involving one or more classifiers" [OMG01]. Classifiers are defined as, "A mechanism that describes behavioral and structural features. Classifiers include interfaces, classes, datatypes, and components" [OMG01]. Activity graphs focus on the sequence of and conditions

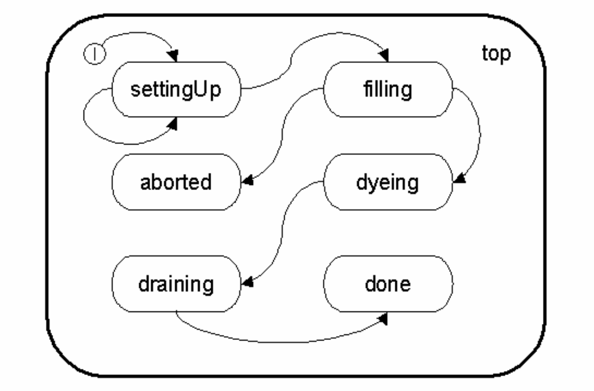


Figure 12. A Simplified ROOMChart for a Dyeing Run Controller [SEL94]

between actions in a process, instead of which classifiers are responsible for performing the actions.

In an activity graph the states are action states that invoke actions and then wait for their completion. The events that can trigger entrance into an action state are:

1. The completion of a previous action state
2. The availability of an object in a certain state
3. The occurrence of a signal
4. The satisfaction of some condition

The major components of an activity graph as described in [FLO99] are as follows:

Activity State: "A state of doing something: either a real-world process, such as typing a letter, or the execution of a software routine, such as a method on a class" [FLO99]. An activity state can be decomposed into sub activities.

Transitions: A transition is a movement from one activity state to the next. It is made up of an event, condition and actions (Although none are required). If no event is included in the transition it is implied that once the activity the transition starts from is finished the transition is triggered. If a transition has a condition placed on it, the condition must evaluate to true in order to allow for the transition to take place. An action is a process that occurs quickly and cannot be interrupted.

Branch: A branch is used to indicate that one of several transitions can be taken. It has one input transition and several output transitions. Every output transition must have a guard.

Merge: A merge has several input transitions and on outgoing transitions and is used to indicate the end of conditional behavior started by a branch.

Forks: A fork has one incoming transition and several outgoing transitions. All of the outgoing transitions are taken in parallel when the incoming transition is triggered. The parallel notation indicates that the order of the activities does not matter, and that the activities may be interleaved.

Joins: A join indicates the end of the parallel activities and is taken after all the incoming transitions have been triggered.

According to Flower, activity graphs are a good tool for workflow modeling and are also useful in understanding how processes work and dealing with multithreaded applications [FLO99]. However, he also notes that they do not make links among actions and objects very clear. Figure 13 illustrates the different components of an activity graph.

2.8.12 EZStrobe/Activity Cycle Diagrams

EZStrobe is a general-purpose simulation system used in the design of construction operations, but is domain-independent. The simulation is based upon activity cycle diagrams and uses the three phase activity scanning paradigm. Although activity cycle diagrams are used in other construction simulations, it is the goal of EZStrobe to be a “very easy to learn and simple tool capable of modeling moderately complex problems with little effort” [MAR01].

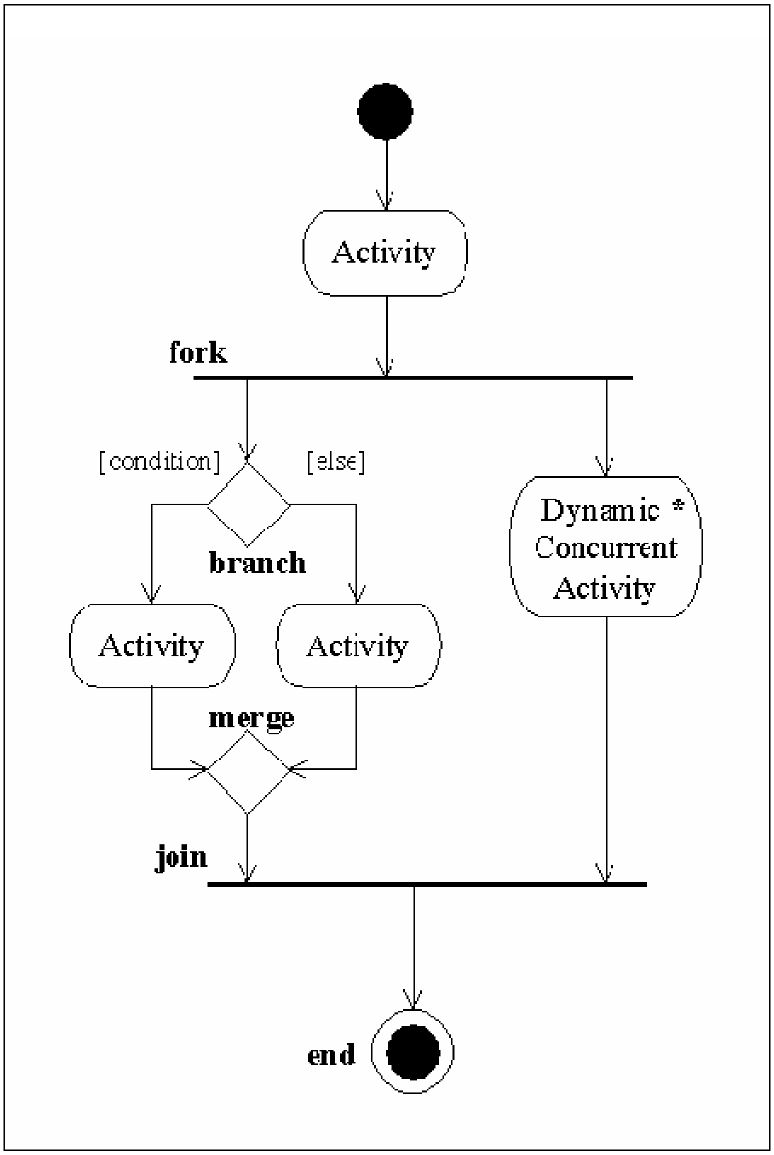


Figure 13. Activity graph [FLO99]

In building the activity-scanning model, the modeler focuses on identifying which activities take place or can take place in a process, what conditions are needed to start each activity, and the outcome of the activity. An activity cycle diagram (ACD) is a graphical representation of the activity-scanning model and consists of a network of circles and squares. In an ACD rectangles represent activities, circles represent queues, and the links between them represent the flow of resources. Figure 14 gives an example of a simple ACD [MAR01].

Queues in an ACD represent idle resources. Resources are placed in the queue by terminating activities preceding the queue and removed by conditional activities following the queue. A conditional activity is an activity that can start whenever the queues preceding the activity have enough resources, while bound activities are activities that are able to start upon the completion of the proceeding activity. Forks are probabilistic routing elements that are used to determine which path is taken when a number of paths are available [MAR01].

There are three types of links that are used to link queues and activities in ACDs. Draw links are used to connect a queue to a conditional activity and show how much of the resource is required in order for the activity to start and how much of the resource is used by the activity. Release links connect activities to queues or a bound activity or fork

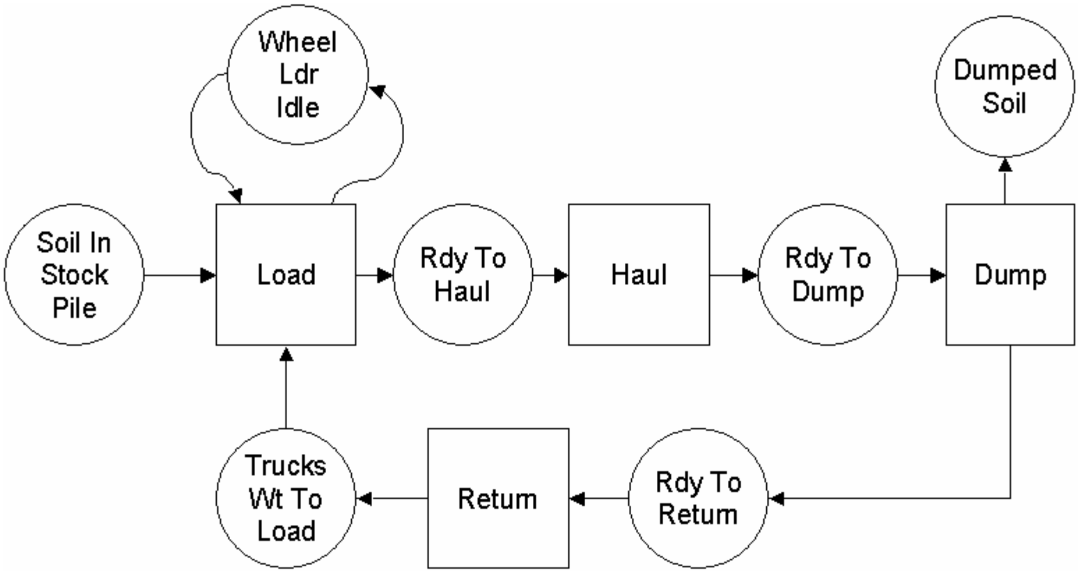


Figure 14. An ACD Diagram for an earth moving operation [MAR01]

and indicate how much of a resource is released by each iteration of the activity. Finally, branch links connect a fork to a queue, another fork, or bound activity and contain the probability that the link will be taken [MAR01].

2.8.13 OPNET

Optimized Network Engineering Tool (OPNET) is a discrete event simulation package used to model networks and is built for the specification, simulation, and performance analysis of communication networks. One of the key features in OPNET is that it facilitates hierarchical model building, allowing for each level of the hierarchy to model a different aspect of the simulation. One such level of the hierarchy used in OPNET is the process model [CHA99].

The process model in OPNET is used to specify the logic flow and behavior of the node models used in the network. Proto-C, which is made up of state transition diagrams, a library of kernel procedures, and the standard C programming language, is used to represent the process models. In order to specify “any type of protocol, resource, application, algorithm, or queuing policy” a state transition diagram approach is used [CHA99]. The states and transitions of a STD are used to define actions of a process in response to certain events, while general logic is specified using the predefined library functions and C code inside the states. The processes also have the ability to create new processes in order to perform subtasks [CHA99]. Figure 15 shows an example of a state transition diagram used in OPNET.

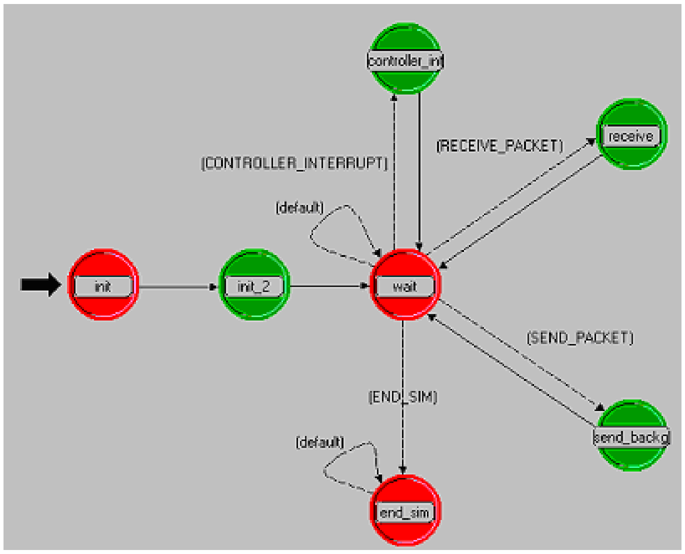


Figure 15. A State Transition Diagram Used in OPNET

2.8.14 Petri nets

As defined by Zimmerman, "a Petri net is a graphical and mathematical modeling tool," consisting of places and transitions [ZIM02]. Places, which are similar to states, are connected to transitions via input arcs, and transitions are connected to places via output arcs. Each place can contain tokens, and the current state of the system is given by the number of tokens (or number of each type of token) that are at each place.

In a Petri net the transitions represent the active part of the diagram and model the activities that can occur in the system. Transitions fire when all their pre-conditions are met, distributing some or all of the tokens in the input places to all of the output places [ZIM02]. Pre-conditions are represented through the number of tokens required in each input place before a transition can fire. Petri nets can be expanded as additional types of arcs can be added to petri nets, and "transitions can be equipped with durations, time intervals, or with stochastic time distributions " [DES00]. Figure 16 shows an example of a Petri net.

Petri nets are often used to represent the behavior of dynamic systems that have multiple objects and events, because Petri nets take into account the dynamic behavior of a system. Furthermore, Petri nets are benefical because they "provide a graphical formalism" and are more precise and formal then data flow diagrams [ALL00].
Jörg Desel recommends using Petri nets for the modeling and simulation of dynamic systems, since "Petri nets provide graphical means for specifying models that support an easy understanding" [DES00]. Furthermore, Petri nets are a desirable model because they have, "a solid mathematical basis and there exists a rich theory on their semantics, their analysis, their simulation and their application in numerous domains" [DES00].

Despite the benefits that Petri nets offer they are not the ideal choice for representing the behavior of individual elements in the system. Petri nets are designed to show the behavior of a system as a whole, and not the states that one component goes through. Furthermore Petri nets can be difficult to understand as it is not always clearly marked how many or what type of tokens are needed for each transition.

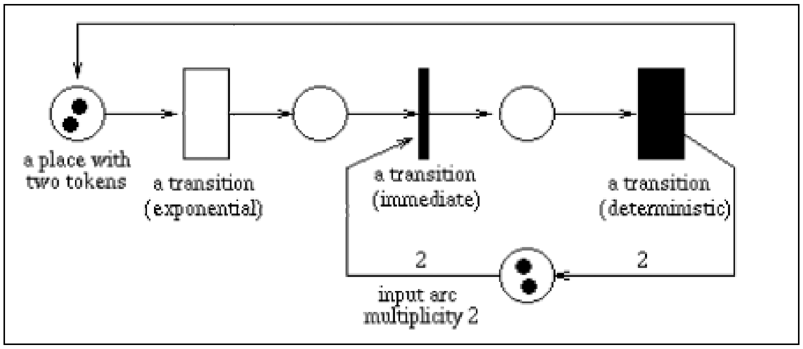


Figure 16. An Example of a Petri net [ZIM02]

2.8.15 Hierarchical Finite State Machines

Nested or hierarchical finite state machines are used in the National Air and Space [Warfare] Model (NASM) to extend composability to behaviors. In NASM, a finite machine framework allows for predicates and actions to be converted into a finite state machine to drive aircrafts decisions in the simulation [PUC00].

A data structure containing a list of states and a pointer to the current state is used to represent a FSM. Each state in the data structure contains a list of predicates, actions, and next states. Predicates are tests that always evaluate to true or false. Actions are activities that are to be performed when a predicate evaluates to true, and the list of next states provides the next state that will occur after a predicate evaluates to true. Each state also contains a process which could be another FSM machine or null [PUC00].

A state machine to determine the flying route of a plane, shown in Figure 17, serves as an example of a simple state machine describing behavior. Predicates test to see if the plane is at a certain point. When the predicate evaluates to true, a GoTo action computes the direction the plane needs to fly to get to the next desired location. Because of the hierarchical nature of the FSM, it would then be possible to divert from and later return to the route at the place where the route was left, without returning to the beginning of the route [PUC00].

The prototype, which was extended to explore the use of composable components in the construction of behaviors, found that there were several benefits to using finite state machines to represent behavior. The first benefit is having the ability to create FSM at runtime, which allows for re-tasking, and for the development of a crude GUI for the design of behavior. The GUI for behavior design is then able to generate text files for the behavior generation. Furthermore, text files were able to be generated from the diagrams, because the predicates and actions define a language with basic elements composed on a simple set of rules [PUC00].

However, the authors neglected to discuss the GUI, and did not state if there ever existed a graphical representation of the state machines created for each behavior, or if the state machines made the behaviors easier to understand. Furthermore, although the article stated complex behaviors could be created from the finite state machines, it did not give a complex example or discuss what happens to the finite state machines as the complexity of possible inputs grows. Additionally, temporal issues such as duration and receiving a message were not addressed.

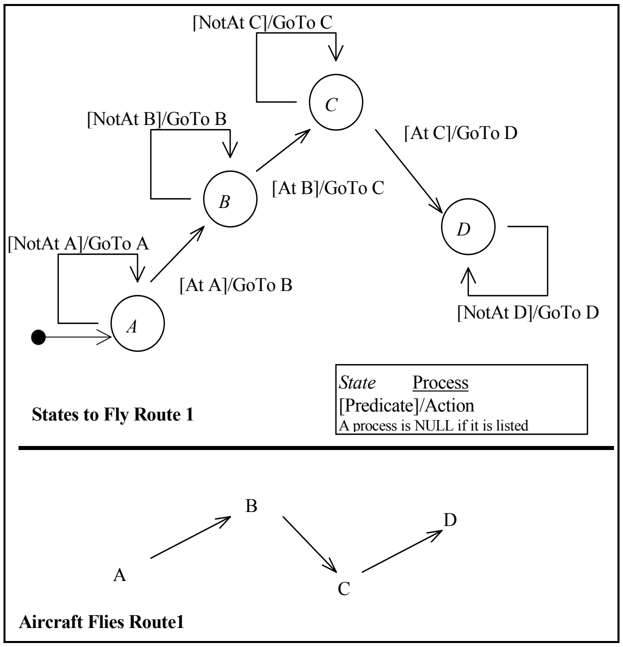


Figure 17. Diagram of a Fly Route Hierarchical State Diagram

2.8.16 Logic diagrams and User Composable Behaviors

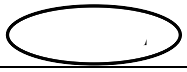
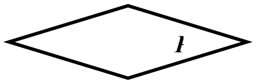
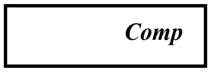
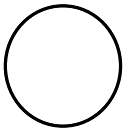
Logic diagrams were used in the Composable Behavior Technologies (CBT) project to allow realistic tactical behaviors to be "easily composed from a set of primitive behaviors" and to allow the end user to create new behaviors that meet her simulation or scenario requirements [COU97]. The purpose of the CBT project is to explore possible methods that would allow users of a Semi-Automated Force simulation system "to create customized behaviors for the simulated entities" [COU97]. In order to allow for the behaviors to be easily composed, a GUI allowing users to build logic diagrams was implemented for the CBT project. The behavior editor allows for the sequencing of primitive behaviors in order to provide the user with a more understandable means for representing behavior then other existing methods such as state diagrams. Table 1 lists the available behavior nodes used in the logic diagrams for the CBT project.

In order to allow for temporal conditions that would impact the behavior defined to be included in the diagrams, the temporal conditions listed in Table 2 are implemented. Different connectors are defined to represent each of the conditions. Figure 18 gives an example of a logic diagram used in the CBT project.

The behaviors defined in the editor are converted into a behavior representation grammar, which contains instructions for the execution of the behaviors. The grammar allows for the hierarchical specification of composite behaviors, the sequence specification of behaviors, and the parameters of these behaviors [COU97].

The study described in [COU97] found several benefits in the use of composable behaviors and logic diagrams. These benefits include the easy representation of

Table 1. Available Behavior Nodes for the Behavior Editor in CBT[VON99]

Behavior Palette Nodes	Description
Begin/Complete	All behaviors must have one Begin and one Complete node. These nodes are represented textually.
	This node represents a primitive behavior. Primitive behaviors are the very basic action that an entity can perform. The primitive behaviors that a user may select are based on the current pane's domain type and echelon level
	This node represents conditional behaviors, which are those behaviors that determine the next path based on the results of the condition. The predicate behaviors available to the user are determined by the current pane's domain type and echelon level.
	This node represents the user-developed complex behaviors. The behaviors available are determined by the currently selected pane's domain type and echelon level.
	This node represents the available communication behaviors for the currently selected pane's domain type and echelon level. This node is comprised of the six different types of communication behaviors. Below each of the types of communications are the actual behaviors that the user may select. The node displays both the type and the behavior name.

hierarchical behaviors that occur concurrently across multiple levels of a military organization and the support of semi-automated behaviors with an explicit representation of user inputs.

2.9 TreeMaps

Treemaps are an interactive visualization method for presenting hierarchical information. Designed by Ben Shneiderman and Brian Johnson, they are described in, *Treemaps: a space-filling approach to the visualization of hierarchical information structures* [JOH91].

Table 2. Implemented Temporal Constraints in CBT [VON99]

Temporal Condition	Description	Proposed

<NULL>	Start execution of behavior as soon as possible. Sequence will be the default condition type.	Sequence
Start Of Other	At the start of another behavior, this constraint will be met and processing will continue. The user selects the other behavior, which is contained within the same composite behavior window.	Start-of-other <behavior>
End Of Other	At the end of another behavior, this constraint will be met and processing will continue. The user selects the other behavior, which is contained within the same composite behavior window.	End-of-other <behavior>
Delay After Other Starts	At a user-specified time after the start of another behavior, the constraint is met and processing will continue. The user selects the behavior and also sets the time.	On-time <time>
Delay After Other Ends	At a user-specified time after another behavior completes execution, the constraint is met and processing will continue. The user selects the behavior and also sets the time.	On-time <time>
On Communication	This constraint is satisfied when a Communication is signaled.	On-order

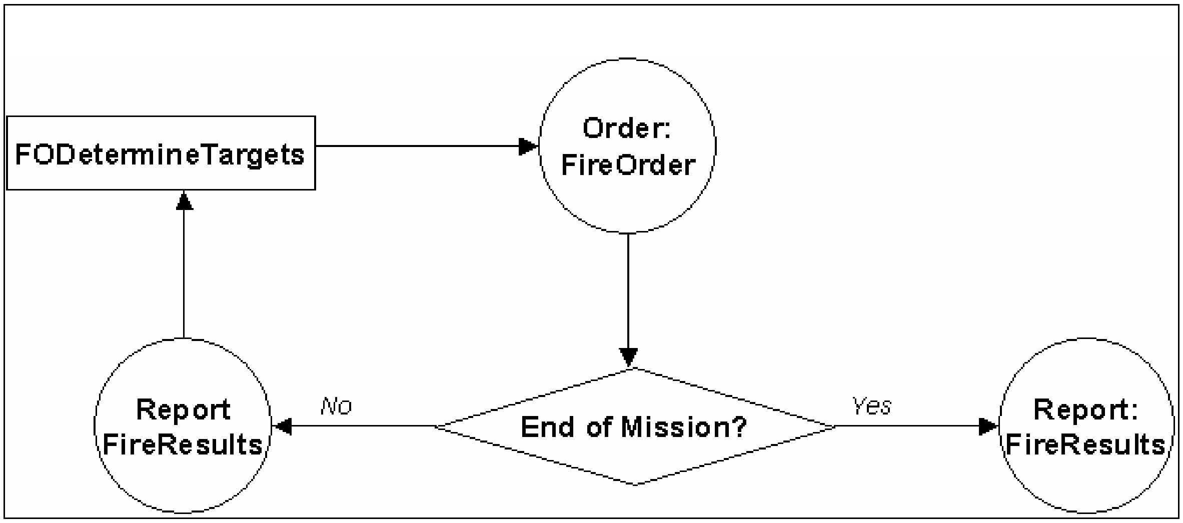


Figure 18. An Example of a Logic Diagram [VON99]

2.9.1 Basic Concept of Treemaps

Traditional, static methods of displaying hierarchies typically make poor use of display space or hide information from users. Treemaps on the other hand utilize 100% of the screen space and allow for information about the hierarchy such as size or force capability, typically hidden from the user, to be displayed. Information about the hierarchy is interactively presented by allowing users to specify the presentation of content and structure.

Treemaps work by partitioning the display space into rectangular bounding boxes that represent the tree structure. Each unit in the hierarchy is placed in the bounding box representing its parent. The higher the importance of the box, the more display space it is allocated. The importance of the bounding boxes are determined by the weight of the nodes inside of the box, while the drawing of the nodes inside the bounding box is determined by the content of the individual nodes. The content of the box determines the weight or importance of the box and can be interactively controlled. Other properties that the user can have control over include colors, borders, etc. Figure 19 shows an example of a treemap used to represent a file structure [JOH91].

2.9.2 Rules for creating a Treemap

The relationship between the structure of the hierarchy and the structure of the treemap drawing as given by Shneiderman and Johnson [JOH91] is as follows:

1. If Node 1 is an ancestor of Node 2, then the bounding box of Node 1 completely encloses, or is equal to, the bounding box of Node2
2. The bounding boxes of two nodes intersect if one node is an ancestor of the other

3. Nodes occupy a display area strictly proportional to their weight
4. The weight of a node is greater than or equal to the sum of the weights of its children

The content of the node can be displayed through visual display properties such as color, texture, shape, border, and blinking. Of these display properties, color is the most important and it can be an important aid to quick and accurate decision making.

2.9.3 Potential of Treemaps for representation of simulation scenarios

As the organization of units and entities in military is a hierarchy, there are several potential benefits of applying treemaps to the representation of troops in simulation scenarios. First, by using treemaps all the units and entities in the battlefield can be displayed on one screen. Through the use of an interactive display the user can

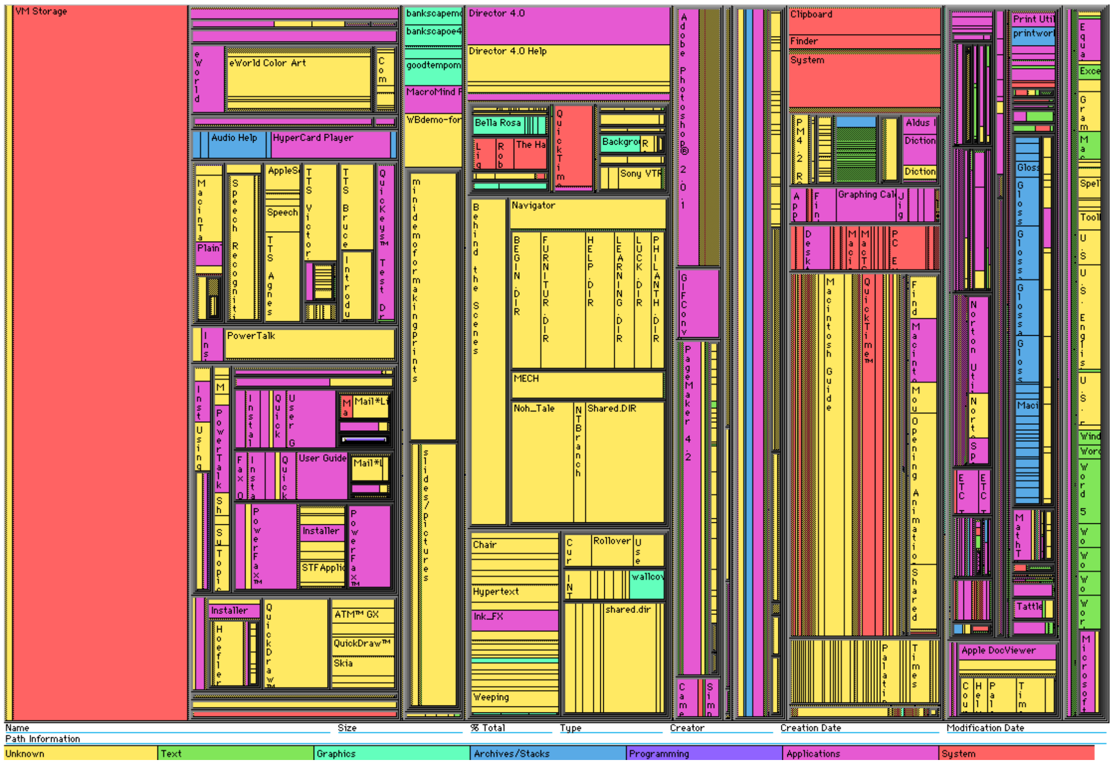


Figure 19. A Treemap of a File Structure [JOH91]

obtain more information about specific units without having to click up and down through large tree structures opening and closing nodes. Furthermore, in treemaps the relative size of the forces to each other is displayed. In several traditional representations of hierarchies each node in the diagram is given equal value regardless of the size of the node. However, in treemaps, the nodes (or entities and units) that are bigger are allocated more space on the screen giving a visual indication of their size relative to the other nodes.

Second, information from the scenario that is hidden or textually displayed in other hierarchically representations can be displayed through display properties of the treemap. Some example properties could be the health of each unit/entity, units/entities that have or have not been assigned missions, and the location of units/entities in the scenario. The information displayed through the treemaps could then be used to facilitate decision making and help in the analyzing the results of simulation scenarios.

2.10 Summary

Composable simulations offer several benefits to the field of simulation and modeling including the potential of lower production costs, greater consistency and validity, quicker scenario development, and greater re-use. Despite the benefits that can be provided by composable simulations, the field has challenges to be overcome. Many of theses challenges result from a lack of key theories in composable simulation and no common specification of components used in composable simulations. One approach to overcoming some of these barriers is to develop a visual language geared toward composable simulations and simulation scenarios.

Visual languages and diagrams have several benefits over pure text in aiding comprehension. The Unified Modeling Language serves as one example of a visual language and is the standard visual language for representing software systems. Although UML has been applied to simulation scenarios in previous studies it, is not ideally suited to describe simulation scenarios.

There are several behavior specification techniques used in software engineering and modeling and simulation. These methods included process graphs, finite state diagrams, extended finite state diagrams, Mealy machines, Moore machines, state charts, process dependency diagrams, SADT activity graphs, ROOMCharts, UML Activity graphs, finite state machines, and logic diagrams.

III. Methodology

3.1 Introduction

Composable simulations potentially offer multiple benefits to the field of simulation and modeling. However, there are many obstacles that composable simulations need to overcome, including a standard specification for simulation scenarios. One solution to this problem is the use of a visual language to represent different aspects of simulation scenarios. This thesis looks at the development of a visual language to represent the high-level behavior of, and hierarchical relationship between, entities in simulation scenarios.

This chapter gives an overview of the methodology used in the research presented in this thesis. First, a background of the problem is presented. Then the objectives of the visual language for simulation scenarios are discussed, followed by the identification of properties of composable simulations that need to be included in a visual representation of the scenarios. Finally the principles of visual languages are discussed and the evaluation criteria used are stated.

3.2 Background

The background for the methodology implemented in this thesis comes from several different areas. Topics discussed in the background include composable simulations, the motivation behind the development of a visual language for simulation scenarios, the differences between software and simulations, and the short comings of behavior specification techniques when applied to simulation scenarios.

3.2.1 Composable Simulation

Composable simulation is based on the premise that the development time of simulations will decrease and accuracy of simulations will increase if they can be created using existing components. There are two aspects in the development of composable simulation: the development of composable simulation environments and the development of scenarios for them. The simulation environment specifies the architecture of the simulation while scenarios are used to specify what should be simulated.

Components in simulations are pieces of code that might represent a model, algorithm, function, or group of models. Each component provides an interface by which it can interact with the simulation system and other components in the system. Components are also designed so they can be used as building blocks, allowing complex components to be built from smaller components. Components can be pieces of software used in the architecture of the simulation model or pieces of code used in the simulation scenario. Ideally, when creating an architecture or scenario a user would be able to select components from a repository and then manipulate them to create the desired result.

3.2.2 Motivation of Applying a Visual Language to Composable Simulations Scenarios

In Chapter 2, several obstacles that need to be overcome in order to make composable simulation a reality were identified. Many of the obstacles mentioned can be related to the fact that there is currently no common descriptor in the development or specification of components among different simulations. The high-level architecture (HLA) is one attempt at standardization is difficult to learn and use. Furthermore, HLA is used to describe simulation systems and not the scenarios used by simulation systems. Scenarios produced by simulations are very different from the simulations themselves and are written in a language specific to the simulation for which they were created.

One method of describing the properties, behavior, and structure of “things” in other disciplines without dependency on the implementation of the “thing” is through the use of diagrams. In software engineering, the Unified Modeling Language (UML) is used to document many aspects of software systems. Similarly, there are several ways diagrams can be applied to the representation of simulation scenarios.

In particular, diagrams can be used to represent the behavior of, and hierarchy between, entities inside simulation scenarios. As presented by Robert Horn in his book on visual languages [HOR98] diagrams have several advantages over plain text. First, diagrams can help the user better comprehend the assigned behavior of components because they “help the learner build run-able mental models” that portray “each major state that each component can be in and the relations between a state change in one component and the state changes in other components” [HOR98]. Furthermore, diagrams represent the hierarchy between the entities in the simulation scenarios better than a textual description because diagrams are more efficient at representing structure and are better at representing complex relationships than text alone.

3.2.3 Differences Between Software and Simulation

As discussed in Chapter 2, although software systems and simulation scenarios have similarities, there are several key differences between them. The main difference is the focus of what is being represented. Software focuses on objects and defining what properties and behaviors an object has. It also looks hat how objects are related to other objects. Simulations, focus on actual instances of objects and what properties, behaviors, and interactions each instance has.

3.2.4 Shortcomings of Current Visual Languages

Currently there are several different representations of behavior specification used in software engineering and simulation. Of the different types of diagrams discussed, Process Dependency Diagrams, which are used in information engineering to represent precedence relations between activities, depict the type of behavior most similar to the missions assigned to entities in simulation scenarios. Process Dependency Diagrams are directed hypergraphs that use nodes to represent on going activities (processes) and edges to represent the transitions between the activities. Three deviations of process dependency diagrams used in modeling are activity graphs, ROOMCharts, and logic diagrams [FLO99], [SEL94], [MCC00]. Activity graphs, a part of UML, are used to "model processes involving one or more classifiers, and focus on the sequence and actions conditions for the actions" [OMG01]. ROOMCharts, found in real-time object-oriented modeling (ROOM), are graphical representations of extended state machines and are used to represent the high-level behavior of entities in real time systems. Finally, logic diagrams used by Composable Behavior Technologies allow users to create complex behaviors from a defined set of primitive behaviors. Chapter 2 gives a description of each type of diagram.

Although the above diagrams sufficiently represent the information they were designed for, each of the diagrams has shortcomings when applied to the modeling of the sequence of activities assigned to entities in simulation scenarios. The following section discusses the properties a language for the representation of high-level behavior of simulation scenarios needs to contain. Then, a discussion of the specific advantages and disadvantages of each of the diagrams when applied to the problem domain is presented.

3.3 Behavior Properties of Entities in High-level Simulation Behavior:

In order to accurately represent the behavior of entities in mission-level simulation scenarios the behavior specification method used needs to be able to represent and support certain properties. These properties include reactions, parameters, temporal conditions, composability, focus on the activities, and a high-level of abstraction.

Reactions: In simulations there are behaviors that are not planned. Instead they occur as reactions to other events in the simulations scenario. For example, in a scenario a plane might be given an ingress command, but upon detecting an enemy plane divert from the ingress and go into an attack activity. There should be a way to indicate that this event is a reaction and not part of the assigned activities.

Parameters: As the activities assigned to the entities represent behavior models that accept parameters, the specification used needs to be able to represent the values that the user entered into the model. Example parameters might be a location, maximum speed, or formation type.

Temporal Conditions: Temporal conditions are used in the behavior specification of entities in simulations and therefore the behavior specification technique used should be able to accurately represent temporal conditions.

Composability: In order to reduce the complexity of behaviors in simulations the user should be given the ability to compose the behaviors into high-level behaviors.

Focus on Activities not on Transitions: The diagram should represent the activities that the entity performs and not the actual state of the entity. Furthermore the transitions in the diagram should not represent data flow as the purpose of the specification is to show the changes in activities that the entity is performing and not mutations of data. Furthermore, as the purpose of the diagrams are to show the activities the entity performs the transitions should only serve to represent the precedence of one activity in relation to another and any condition that must be met before the transition can take place.

Higher Level of Abstraction: The user is not concerned with how the behavior is implemented, but instead they care about the high-level description of the behavior assigned to entities and the parameters that they can modify. Therefore the specification should provide high-level detail without touching how the behavior itself is implemented.

3.4 Drawbacks of the Current Behavior Specification Techniques

Although the diagrams discussed in Chapter 2 adequately represent the data they were designed to represent, each of the diagrams has both advantages and disadvantages when applied to the representation of the high-level behavior of battlefield entities in mission-level simulation scenarios. The following sections discuss some of the major benefits and drawbacks of several of the diagrams mentioned in Chapter 2 when applied to the representation of such behavior.

3.4.1 ROOMCharts

When applied to the type of behavior this thesis is looking at, ROOMCharts offer several benefits. First, since ROOMCharts represent the high-level behavior of actors, they relate very closely to the high-level behavior of entities in simulation scenarios. As a result of this similarity they serve as a good starting off point for the representation of the behavior of entities in simulation. Second, they implement the concept of transition between the nodes as the changing of activities performed by the actor and allow for guards to be placed on the transitions. This is the same representation of transitions that needs to be included in the behavior specification used. Furthermore, in order to reduce the complexity of the diagrams, the transitions can be labeled and defined separately from the chart.

Despite the benefits of ROOMCharts they have some disadvantages when applied to the domain. First, the transitions in ROOMCharts are triggered by the arrival of messages to the actor and not the completion of the activity currently being executed or the condition on the transition evaluating to true. Second, the chart includes the extra notion of actions or tasks attached to transitions, which are not needed for the behavior this research is trying to represent.

3.4.2 Activity Graphs

Activity graphs offer several benefits to modeling the problem domain in that they are used to model the activities of a process, which is very similar to the modeling of a sequence of behaviors assigned to an entity. Furthermore, transitions are used to represent the movement from one activity to the next, and are invoked by triggers or the completion of the action in the activity state the transition starts in.

Activity graphs, however, have disadvantages in that they model a process, rather then the behavior of one entity. In the execution of the process several classes can be included, but the behavior the specification required for this research is scoped down to a single entity. Furthermore, the actions represented by the diagrams are different than activities performed by entities in simulations in that actions are placed as entrance and exit actions in the diagram, occur quickly, and cannot be interrupted. Activities performed by simulation entities, however, can be lengthy and can be interrupted. Finally, the graph allows for concurrent behavior which is not needed for the domain being modeled. Concurrent behavior happens in activity graphs when two or more activities are being executed at the same time in a particular process. In the high-level behavior of entities in simulation scenario, an entity can only be performing one activity at a time.

3.4.3 Activity Cycle Diagrams

The main benefit of activity cycle diagrams is that the nodes in the diagram represent activities and are the focus of the diagram. However, activity cycle diagrams are not ideal for the purpose of this thesis because they focus on the flow of resources through a process, rather then a sequence of events. Therefore the conditions on transitions are based around the availability of resources, rather then the completion of a task or the evaluation of some condition to true.

3.4.4 OPNET

The main benefit of the state transition diagrams in OPNET is that the diagrams are used to model the actions of a process, which is similar to the sequence of behaviors of entities. However, OPNET relies on predefined library functions and a variation of the C programming language in order to truly represent the behavior of the process. Furthermore it represents the actions of a process on an entity and not necessary the actions an entity takes. Also, the state transition diagrams used in OPNET are modeled to have several entities flow through the same process. In the behavior this research specifies, different entities of the same type may perform two different sets of actions. Finally, the STDs in OPNET use events to trigger transitions, and OPNET does not provide a way for guards to be placed on the transitions.

3.4.5 Petri Nets

Petri nets are very beneficial in simulations in that that they have a solid mathematical basis, which aids in the accurate execution of the model. Furthermore, when applied to the representation of behavior of entities they allow for time intervals and durations to be placed on the transitions.

However, Petri nets are not ideal for the behavior representation needed for this research because the transitions represent activities and the states are used to represent the conditions for triggering the transitions. This representation is opposite of the way most process dependency models work, making petri nets more difficult to learn for users of other diagrams. Furthermore the triggers on transitions are represented by tokens, which does not facilitate reading of the model. Finally, Petri nets are designed to model the behavior of a system and the behavior of entities moving through the system, which is quite different than the behavior of individual entities.

3.4.6 Hierarchical Finite State Machines

The hierarchical finite state machines used for NASM have several benefits when applied to the problem domain. First, they allow for composition by allowing state machines to contain other state machines. Second, they contain predicates which must evaluate to true in order for the actions to be performed. These predicates are similar to the conditions on the transition triggers that are needed to represent temporal conditions. Third, the activities represented by the states in the finite state machines are similar to the activities performed by entities in the scenarios. Finally, the rules of the finite state machine allow text files to be generated from state machines.

Despite the benefits of hierarchical finite state machines there are some drawbacks. First, in the applications it is being applied to, the level of detail is a lower level of abstraction then what this research seeks to define. Second, in order for an activity to take place a predicate must evaluate to true, which is different from the concept of activity completion found in the problem domain. Also, different activities are performed based on the predicate. In the problem domain, the predicates are mainly used to signal when an activity will happen and not what activity will happen. Finally no abstract syntax for the diagrams was given, making it hard to picture what the actual diagrams look like.

3.4.7 Logic Diagrams in Composable Behavior Technologies

As with hierarchal state machines, logic diagrams offer many benefits to the representation of high-level behavior of entities in simulation scenarios. Many of the benefits come from the fact that logic diagrams are used to represent the sequencing of primitive behaviors, which are very similar to the type of behaviors this research is trying to represent. In addition to having activities and connectors between the activities, the logic diagrams also allow the representation of user-composed behaviors and the specification of user-defined parameters in the activities. Both of these capabilities are required for accurate representation of the problem domain. The study that used the logic diagrams also validated the use of the diagrams because the benefits that came from using the logic diagrams further demonstrated the validity of using diagrams to represent behavior.

Despite the benefits that the logic diagrams used in CBT have to offer, they also contain drawbacks, as the behavior represented by logic diagrams in CBT is different from the behavior being represented in this research. Instead, the logic diagram allows users to compose primitive behaviors into more complex behaviors. The behaviors defined by CBT are at a lower level of abstraction and require greater domain knowledge than the behaviors defined by this research.

As a result of the differences in the behavior being represented by logic diagrams and the behavior being researched, several disadvantages to logic diagrams arise. First, temporal conditions are implemented through a defined set of connectors or transitions. As a result of defining a limited number of connectors only a few temporal conditions have been defined and can be used in CBT. Second, some of the temporal conditions deal with concurrent activities, which are not allowed in the specification of a behavior of a single entity. Third, the predicates tend to lean toward only two options, yes or no. For the behavior being studied by this research it is necessary to allow for more then two possible transitions to be modeled. Fourth, several different shapes are used to represent states and behavior, which makes the diagram more difficult to read than other variations of process dependency diagrams. Finally, some of the components in the logic diagrams are included because they are needed for the lower level of detail not addressed by the problem domain of this research. Therefore, they should not be included in the developed behavior specification in order to reduce the complexity of the language.

3.5 Design Objectives for the Visual Language for Simulation Scenarios

The overall goal of the work presented in this thesis is to create a visual language that aids in the comprehension and building of simulation scenarios. In particular, the language is intended to aid in the development and comprehension of composable simulation scenarios at the mission-level by allowing for easier comprehension of the behavior of, and hierarchical relationships between, entities in the scenarios. The language developed has the ability to be applied to multiple mission-level simulations. By changing how the components of the language can be connected and defining required attributes for the components, the diagrams will reflect the architecture and limitations of the simulation environments the scenario they describe were created for.

By modeling the components of simulations and their scenarios it is the hypothesis of this thesis that the visual language developed will increase the understanding of the structure and behavior of the entities in simulation scenarios.

Furthermore, having a common language to describe the simulation scenarios serves as a basis for a tool that could potentially allow for one scenario to be generated for multiple simulation platforms, or the conversion of a simulation scenario from one platform into a simulation scenario for another platform.

For the purposes of this research, the main objective has been scaled down into two sub-objectives. The first sub-objective is the development of a visual language that describes the assigned behavior of components acting as entities in simulation scenarios. The second sub-objective is the application of treemaps to the hierarchy between entities in mission-level simulation scenarios.

3.5.1 Representation of Assigned Behavior in Simulations

The scope of the first objective has been reduced to apply to the representation of missions for entities in mission-level simulations. In mission-level simulations a mission can be defined as a sequence of activities performed by entities in the battlefield. Entities range from a single individual to an entire battalion. In a mission for an entity there may be conditions between the sequence of two events and alternate courses of action invoked by the conditions of the environment referred to as reaction tasks. By creating a visual language to describe the sequence of behaviors, the visual language aids in the comprehension of already completed scenarios and the in the creation of new scenarios.

3.5.2 Application of TreeMaps

The second sub-objective of the research conducted was to increase the understanding of hierarchies in scenarios. This was completed by applying the information visualization technique known as treemaps to the hierarchy of entities in simulations scenarios. By applying treemaps to the hierarchy of entities in battlefield simulations scenarios users of the system can look at all the entities in the battlefield in a single space, without having to trace through long list of units or maximize/minimize nodes of a tree. Furthermore, by customizing the properties of the treemap such as color and the borders of the units in the hierarchy the user is given the ability to customize the treemap to present the desired information in a way that assists user comprehension.

3.6 The Visual Language

In order to meet the objectives stated above, diagrams named Simulation Behavior Specification Diagrams (SBSD) are specified and described in Chapter 4. Also in Chapter 4, treemaps are adapted to accurately display the information stored in the hierarchy of entities in simulation scenarios. When designing the visual language for the SBSDs and in the adaptation of the treemaps several different properties of simulation scenarios were taken into account.

3.6.1 Properties Needed By the Visual Language

Due to the domain the visual language is being applied to there are several properties that the visual language should be able to represent. First, the visual language must allow for the specification of attributes in the representation of activities. This requirement is necessary as the user must be able to specify certain attributes of the behaviors assigned to entities. Next, as temporal conditions such as duration of a task, reaching a control point, or receiving a message from another entity can determine the completion of one behavior and the start of another, the diagrams must allow for constraints to be placed on transitions between two activities. Another desirable property of the visual language is the ability to compose a sequence of activities into a new activity. By allowing activities to be composed it allows for the user of the diagram to pick the level of detail they want to view. It also saves time as the user does not have to repeatedly assign the same sequence of activities multiple times.

Furthermore, some of the activities an entity performs in a mission-level model are reactions to events, rather then the originally planned sequence of activities. Therefore, the language should provide a visual indicator of what transitions are reactions, and if the transition indicates a permanent or temporary deviation from the original path. The language should also be capable of forward and backwards generation. In forward and backwards generation, the scenario representation used by the simulation can be accurately expressed by the visual language. Furthermore, the representation of the scenario by the visual language should be able to be converted into a scenario file used by simulations.

Finally, the language should be able to efficiently represent large groups of entities, information related to the entities, and hierarchy of the entities in the simulation. In a mission-level simulations there may be hundreds of entities. Therefore, the diagrams should be flexible enough to allow the user to select which entities and on what aspect of the entities the diagrams created by the entity focus on.

3.6.2 Areas of Visual Languages not Addressed

The diagrams presented in this research only address a subset of the information that would need to be represented in order to completely represent simulation scenarios. Specifically the diagrams do not address the architecture or definition of the entities used in the simulation. Nor do the diagrams express communications or relationships between the entities outside of the hierarchical command structure of the entities. Finally the diagrams presented do not address the structure of the simulation systems. These areas, which are important and are needed to completely create a visual language for simulation scenarios were not addressed in order to reduce the scope of the work and are left for future investiagations.

3.7 Evaluation Parameters

The visual language developed for the behavior specification of entities in the scenarios of mission-level models is evaluated qualitatively in Chapter 5 on how well it adheres to the principles of modeling languages. Simplicity, uniqueness, consistency, seamlessness, scalability, supportability, reliability, and space economy are all key principles in the design and evaluation of visual languages [BRO00].

3.7.1 Principles of Modeling Languages

Simplicity: Simplicity is the idea that a simple language can be fully understood by the user and therefore allows the user to discover deficiencies in the language and makes the user better-equipped to handle complex tasks. Furthermore, in simple languages the user knows how to use the language in simple ways and therefore will most likely know all the consequences of combining the language features. Simplicity is the most important principle, because without simplicity none of the goals of a modeling language can be reached. Furthermore, disadvantages of complex languages include a large overhead in learning the language before it can be used and difficulty in the implementation of tools supporting the language.

Uniqueness: Uniqueness is the property that a language “provides one good way to express every concept of interest, and it avoids providing more then one.” A language that has the property of uniqueness is smaller and more explainable than one that has duplicate features.

Consistency: A language that has consistency is a language that has a purpose and any feature in the language that does not support the purpose is discarded. Consistency of the language should not be confused with consistency among the models of a language, which is more of a reliability issue.

Seamlessness: “Seamlessness allows the mapping of abstractions in the problem space to implementations in the solution space without changing notation, thus avoiding the impedance mismatches that often arise throughout the development process” [BRO00]. For the domain of this research seamlessness deals with the behaviors of the scenarios being represented by the visual language without having to change the meaning of the notation or add new notation. Not changing the meaning or notation of the language prevents errors when going from the visual representation of the scenario to the creation of the scenario file.

Scalability: Scalability is the property that a language is useful for both big and small systems. To be scalable a language must provide a concise mechanism for describing fundamental abstractions of the problem domain, allow the details of abstraction to be hidden, and provide a grouping mechanism so the modeler can “collect abstractions, name them, and hide their details” [BRO00]. As one of the goals of the visual language designed for this research is for it to be composable, the language needs to be able to represent both simple and complex sequences of activities.

Supportability: Supportability has two aspects. First, since models are used by humans for writing or drawing models, and this is often done on a white board or with pencil and paper, the language should be easy to produce by hand. Second, for large software systems there should be tool support for drawing and managing the models as well as maintaining the system as the development process proceeds. Therefore the notation syntax should be easy to draw and display on a computer screen and the semantics "should be defined to that it can be automatically or semi-automatically translated into code" [BRO00]. For this research the diagrams created by this language should be able to be converted into scenario files through development tools, but should also be easy to draw by hand.

Reliability: A language that satisfies the principle of reliability is one that meets specifications and reacts appropriately when given unexpected or incorrect input. The idea of quality is directly supported by reliability.

Space economy: The principle of space economy states that "models should take up as little space on the printed page as possible" [BRO00] Space economy is important in the representation of behavior of simulation scenarios because the user may want to view the behaviors of several different entities at once. Furthermore, if a behavior is complex, the user should not have to flip through pages to view the entire behavior.

3.8 Evaluation Criteria

In order to test if the visual language meets the evaluation criteria, case studies of the visual language were evaluated using simulation scenarios from the OneSAF simulation. OneSAF is an experimental, composable simulation used by the Army for the education and training of battalion officers. In OneSAF, users are given the ability to assign a sequence of tasks to battlefield entities. Furthermore, the battlefield entities in the OneSAF simulation are arranged in a typical command hierarchy, making it an ideal candidate for the application of a treemap.

The following criteria are based on the evaluation criteria used by Hakan Canli in the evaluation of a the general modeling language presented in [CAN02], and the work done by van Harmelen, Aben, Ruiz, van de Plassche in [HAR96].

Expressiveness: Were certain aspects or properties impossible to express? If so, what? Were some things difficult to express?

Frequency of errors: What are the most common errors and the frequencies of those errors. Why did those errors occur? How can they be avoided? Where is the potential for errors?

Redundancy: Was redundancy present in models? Is it possible to identify different types of redundancy? How can redundancies be avoided? Where did the redundancy occur?

Locality of change: Do changes propagate through the models? If so, what are the causes, and can they be avoided?

Reusability: Do the models enable reusability?

Reliability: Do models enable consistency checks? If not, why and how can the inconsistencies be avoided?

Translatability: Are the models consistent and expressive enough to be used as an input to a simulation tool?

Compatibility: What is the distribution of results of the above criteria?

3.9 Summary

Composable simulations allow developers to compose simulation scenarios through the use of components. Currently there is no common specification for the scenarios of composable simulation. Visual languages are one type of specification used by other fields, such as software engineering, to model objects and systems without regard to how they are implemented. In order to accurately represent the behavior of the domain being researched several properties that a language must have were identified. These properties include the ability of the language to represent reactions and temporal conditions. Furthermore, the components representing activities must be able to accept parameters and the language should be composable. Finally the language should focus on activities and work at a high level of abstraction.

Several different behavior specification techniques are used in software engineering and simulations. These specifications include ROOMCharts, activity graphs, activity cycle diagrams, state transition diagrams, Petri nets, hierarchical finite state machines and logic diagrams. Each of these specifications has advantages and disadvantages when applied to the problem domain.

Good modeling languages have several different properties. These properties include simplicity, uniqueness, consistency, seamlessness, scalability, supportability, reliability, and space economy. In order to evaluate if the language developed has these properties, case studies are conducted in Chapter 5 using scenarios for the OneSAF simulation. Through the case studies the language is evaluated on its expressiveness, frequency of errors, redundancy, locality of change, reusability, reliability, translatability, and compatibility.

IV. Language Definition

4.1 Introduction

In order to make simulation scenarios easier to comprehend, easier to build, and to provide a common descriptor for scenarios from multiple simulations, this research proposes a visual language for describing simulation scenarios. The visual language focuses on the representation of behaviors assigned to entities in scenarios and the hierarchical relationship between the entities in the scenarios. This chapter presents both aspects of the visual language. First, simulation behavior specification diagrams used to visually represent the high-level behavior of entities in simulations, are defined. Second, the application of treemaps to simulation scenarios, which is designed to help the user understand the hierarchy of the entities in simulation scenarios, is presented.

4.2 Simulation Behavior Specification Diagrams (SBSD)

Simulation behavior specification diagrams describe the sequence of activities assigned to battlefield entities in simulation scenarios. The use of the diagrams aids in the comprehension and composition of behaviors assigned to entities in simulation scenarios.

As the simulation behavior specification diagram is designed to be used by several different simulation scenarios, the language can be broken down into two parts. The first part of the language defines the base components, syntax, and semantics of the language while the second part is the specification of the semantics of the language for scenarios of specific simulation models. By allowing the components of the language to be extended or refined with defined rules for each simulation, the exentsions allows for the details necessary for execution of the simulations to be included in the diagrams without limiting the diagrams to scenarios from one simulation model.

4.2.2 Components

The design of SBSD is derived from the principles of process dependency diagrams and extended finite state machines. As with process dependency diagrams and extended finite state machines, the components of the language can be broken down into nodes and transitions. Figure 20 gives an overview of the abstract syntax of SBSD, using standard UML notation. The abstract syntax shows the components that make up the language and how the components are associated.

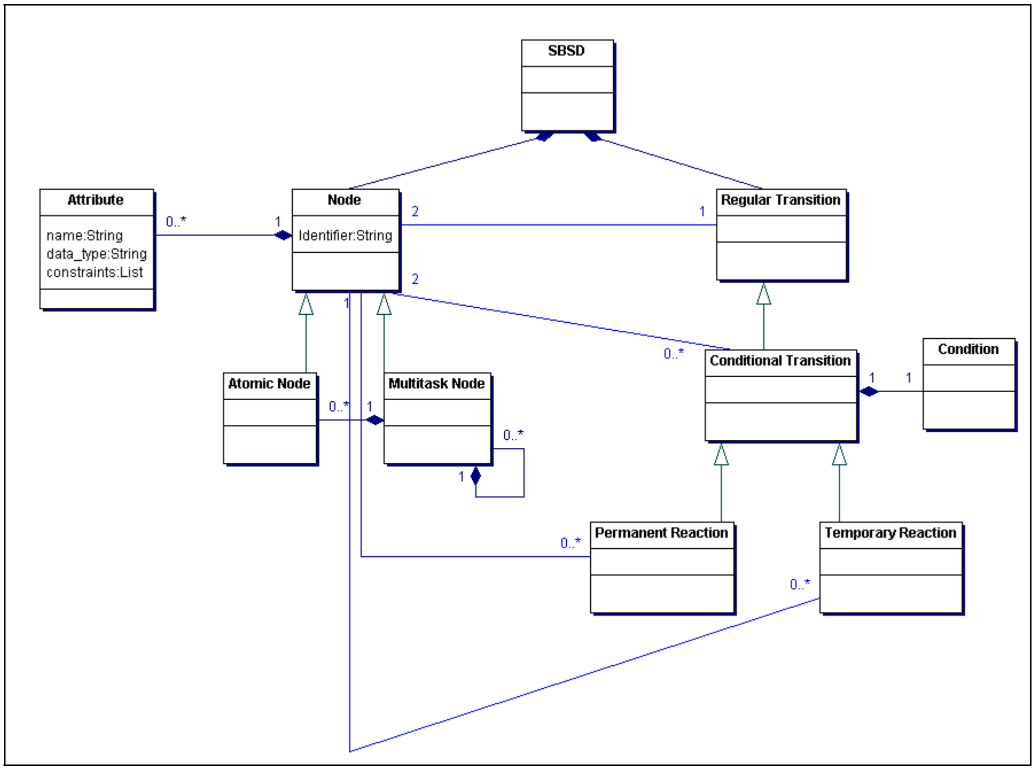


Figure 20. An Abstract Syntax of Simulation Behavior Specification Diagrams

The following sections describe each of the components of SBSD pictured in Figure 20. Based loosely off of the UML specification, each section will consist of several subsections. First, a short description of the component is given. Then the syntax and semantics of the component are stated. Following the syntax and semantics of each component, the purpose of the component in SBSD is discussed, along with a justification for the visual representation of the component. If a subsection is not applicable to a component it is not included. Pictures of the components and sample SBSD diagrams are also given throughout the following sections.

4.2.2.1 Nodes

Nodes represent an activity or a sequence of activities, performed by battlefield entities in simulations. They are the equivalent to states in a state transition diagram or process dependency diagram.

Syntax

Nodes contain one identifier and zero or more attributes. The identifier serves as a description of what activity the node is representing. Attributes represent the user modifiable properties of the node or properties necessary to the execution and comprehension of the scenario. Transitions start and end in nodes. There are two types of nodes in SBSDs: atomic and multi-task node. Figure 21 shows an example of each type of node.

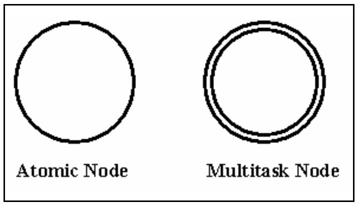


Figure 21. Visual Representations of an Atomic and Multi-task Node

4.2.2.1.1 Atomic Nodes

An atomic node represents a single activity performed or assigned to an entity in a simulation scenario. An atomic node and its attributes are the lowest level of detail available to developers of scenarios.

Syntax

Atomic nodes are entered through transitions. They are exited upon completion of the activity. The end of the activity is either defined by the behavioral model the activity is representing or by the guard condition placed on the transition leaving the node. For certain activities such as a move activity the parameters specified by the user such as a location may be used to determine the end of the activity. An atomic node is represented in SBSD by a circle with a solid border. The attributes of the node can be represented below the identifier, or can be defined separately from the diagram.

Semantics

An atomic node represents the activities performed by battlefield entities in a simulation scenario. The activities represent complex behavior models that have been modeled by domain experts and programmed by software engineers. Depending on the simulation being represented, the behavior may be a complex behavior composed of several concurrent sub-activities. However, for the developer of the scenario, the activity represented by an atomic node is the smallest piece of behavior the developer has control over. The attributes of the node serve as parameters into the behavioral model and allow users to specify information used by the behavior model in the execution of the scenario.

Purpose in the language

As the intent of the diagram is to represent the sequence of activities assigned to or performed by entities in the simulation scenario, atomic nodes are a necessary component of the language. Attributes are included in the atomic node to allow for customization of the behavior models they represent. By allowing for the nodes to have attributes, the node can represent multiple activities, instead of redefining the node for the same activity each time one of the input parameters changes. The attributes also allow for scenario creation as the attributes provides a way for the diagrams to represent the information necessary for simulation execution.

Justification for Visual Component

The circle is similar to an oblong which is the standard visual components used in process dependency diagrams to represent activities. The circle is also the standard representation of states in state transition diagrams. By using the same component it allows for users of process dependency diagrams to be able to read SBSD without having to completely learn a new language.

4.2.2.1.2 Multi-task Nodes

A multi-task node represents a sequence of atomic and multi-task nodes connected by regular and conditional transitions.

Syntax

Multi-task nodes are entered through a transition into the first node of the sequence and are exited through the last node of the sequence through regular and conditional transitions. There can only be one exit transition from the sequence of multi-task nodes.

Through tools that provide an interactive GUI, users can expand or collapse the missions. A condensed multi-task node is represented by a circle with a double line black border. As shown in Figure 22, an expanded multi-task node is represented by the SBSD diagram of the nodes and transitions it contains in the multi-task node placed inside a rectangular box.

Semantics

A multi-task node represents a sequence of activities grouped together to create a more complex activity. The activities represented by multi-task node are one abstraction level higher than activities represented by atomic nodes. Multi-task nodes are used to group a sequence of tasks commonly performed together. By grouping the sequence of tasks, a higher level of abstraction is provided and reuse occurs because the user only has to define a particular sequence of tasks once.

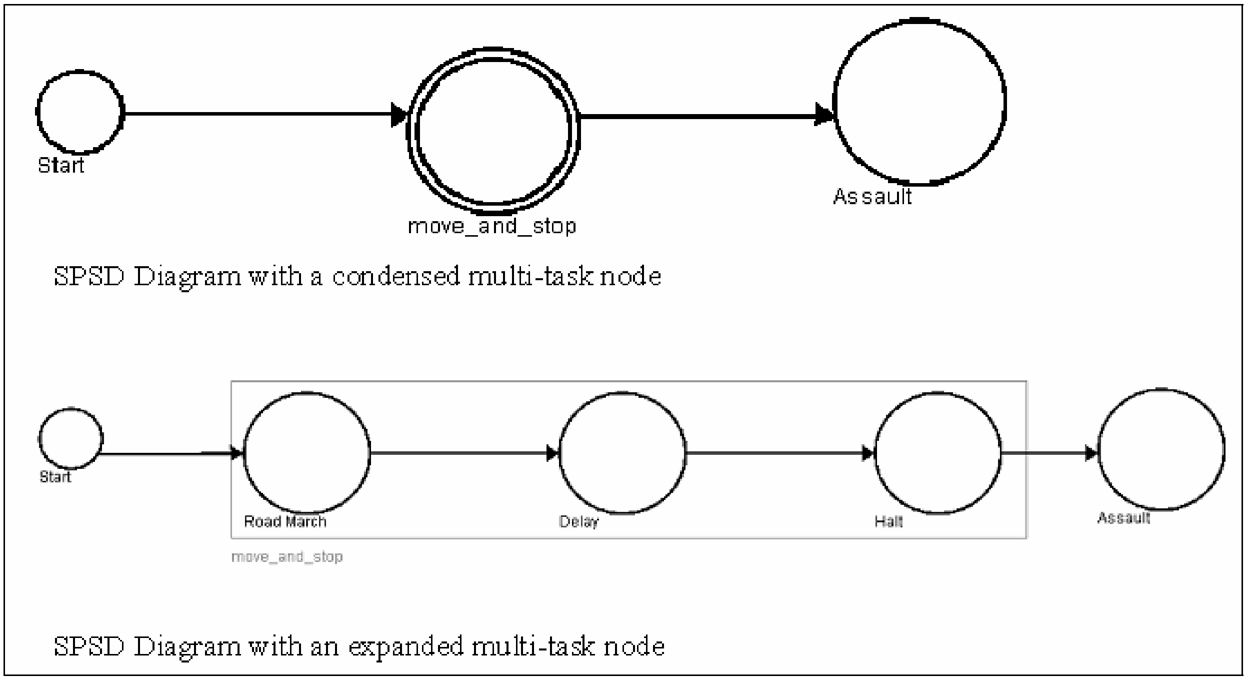

Start
move_and_stop
Assault
SPSD Diagram with a condensed multi-task node
Start
Road March
Delay
Halt
Assault
move_and_stop
SPSD Diagram with an expanded multi-task node

Figure 22. An SBSD diagram with a Multi-task Node

Purpose in language

Multi-task nodes are included in the language to allow for the combination of activities into a higher-level activity. Allowing activities to be combined into one node provides scalability, re-use, and economy of space in the language. Scalability is provided because the multi-task node allows users to combine simple sequences into more complex sequences without resulting in an overly complex diagram. The multi-task node supports re-use because it allows a sequence of tasks to be defined once, and then re-used as necessary. The multi-task node also provides space economy since it takes much less room on the page than the corresponding full sequence of activities. Through software tools, the multi-task node can be expanded and collapsed in order to provide the desired level of detail. In text documents, the multi-task node can be defined in one place and then referred to as needed. The multi-task node differs from an atomic node composed of several concurrent behaviors because it can be defined by the user and represents a sequence of activities rather then a set of concurrent activities.

Justification for visual component

The multi-task node is graphically similarly to the atomic node in order to indicate that an activity is being represented. The notation is slightly different from the notation of an atomic node to visually indicate that the node represents a sequence of tasks.

4.2.2.1 Attributes

Attributes represent the user modifiable parameters of nodes.

Syntax

Attributes consist of a name, data type, and constraints. The name serves as an identifier for the attribute. The data type identifies what type or unit of data is represented by the attribute. An integer, the unit miles per hour, or object could all be data types. The constraints specify the possible input values for the attribute. A constraint can be a range of values, a rule, set of rules, or an enumeration of values.

Semantics

The attributes of a node serve as an interface to the node, allowing users to change or view the properties of the node represented by the attributes without knowing the implementation or detailed behavior of the node. During execution, certain attributes of the node can be updated by the system. Other attributes are not updateable, but serve as parameters defined by the user, that are used by the simulation for the execution of the behavior.

4.2.2.2 Transitions

Transitions are used to specify movement from one node to the next.

Syntax

Every type of transition is composed of a starting and ending node. The starting node specifies which activity the transition is moving from, while the ending node specifies which activity the transition is moving to. A transition can have only one starting and one ending node. The starting and ending nodes must be two different nodes (Although each node can represent the same type of activity).

There are four different types of transitions defined in SBSD. These transitions are regular, conditional, temporary reaction, and permanent reaction. Temporary reaction and permanent reaction transitions are extensions of conditional transitions. Figure 23 shows the visual representation of each of the transitions.

4.2.2.2.1 Regular Transition

A regular transition is used to connect an atomic or multitask node to another atomic or multitask node.

Syntax

The use of a regular transition indicates that upon completion of the activity represented by the start node of the transition, the execution of the activity represented by the end node of the transition will begin. The only associations a regular transition has are its starting and ending nodes. A regular transition is represented by a solid arrow connecting the starting and ending nodes.

Semantics

A regular transition represents the progression in the activity that an entity is performing from the activity represented by the start node to the activity represented by

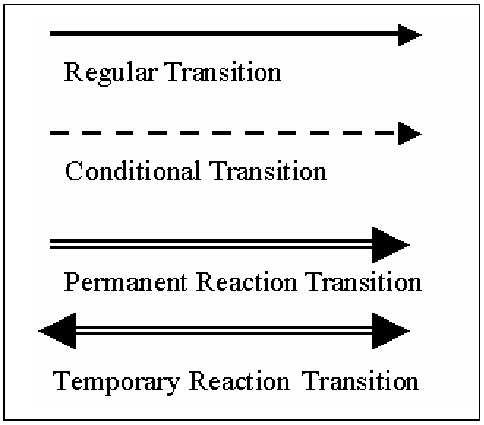


Figure 23. Visual Representations of Transitions in SBSD

the end node of the transition. In activities connected by regular transitions, the end of the activity is defined in the implementation of the activity or behavior model represented by the node.

Purpose of component in language

The purpose of the regular transition is to represent the progression of the entity from completion of one activity to the beginning of the next activity, which is a necessary part of the diagram in order to be able to represent behavior of entities in simulation scenarios. By separating the regular transition from a transition where a condition has to be fulfilled it allows for the diagram to give the user a visual indication that the entity proceeds directly from one activity to the next and that the behavior defines what signifies the end of an activity.

Justification for visual component

The directed arrow was selected to represent transitions as it is the standard representation of transitions in process dependency diagrams.

4.2.2.2.2 Conditional Transition

Conditional transitions represent transitions that contain a guard, or condition that must be met before the transition can take place.

Syntax

Conditional transitions are regular transitions that have a guard assigned to them. A guard is a Boolean condition that must evaluate to true in order for the transition to occur. When conditional transitions are used, the end of the start activity is determined by the guard in the transition and not by the activity represented by the start node of the transition. A conditional transition is represented by a dashed arrow connecting the starting and ending node. The condition is placed in a box next to, above, or below the transition. In the case that the condition is lengthy a label can be placed in the box and the condition defined in a separate location. Through a GUI, the user may also be allowed to minimize or maximize the condition depending on the desired level of detail.

Semantics

Conditional transitions are used to represent the transition of the entity performing one task to performing another task after some event has taken place or a condition has been met. Furthermore, the use of guards on the transitions allow for temporal conditions to be represented in the language. Examples of temporal conditions include the specification of a duration of time the entity performs an activity, an entity passing a control point, or an entity receiving a message. When being applied to specific simulations models each of the models will need to define the type of guards that are allowed in scenarios for that model.

Purpose of component in the language

One of the major drawbacks of some process dependency charts is that they cannot represent temporal conditions, which are an important part of simulation scenarios. By allowing for conditions to be placed on transitions the temporal conditions can be represented in SBSD.

Conditional transitions were made a separate component from regular transitions in order to give the user a visual cue that the transition is not a regular transition without having to study the details of the diagram. By knowing that the transition is a conditional transition, the diagram also tells the user that the condition placed on the transition defines the completion of the activity. In this case the user cannot assume that the start activity was completed as defined by its attributes. For example if a duration of two hours is placed between a move activity and halt activity and the speed of the entity did not allow the entity to reach its destination in two hours, the move activity would not be completed as defined by its attributes. Finally, through conditional transitions, the condition can be defined elsewhere, but still be noted on the diagram, reducing complexity.

4.2.2.2.3 Reaction Transitions:

Reaction transitions represent the transition to nodes that represent activities that serve as reactions to events in the simulation environment.

Syntax

Reaction transitions contain a guard which indicates what events or conditions cause the transition to be taken. Temporary reaction and permanent reaction are two types of reaction transitions defined in SBSD. Reaction transitions are associated with a start node, an end node, and a guard.

Semantics

In simulations, entities may take a course of action different from the one assigned as a reaction to events in the simulation. The change of course in action may be caused by the simulation program or by a user monitoring the scenario as it runs. When the entity diverts from performing the activities it has been originally assigned, it leaves the activity that it is currently executing before its completion, in order to execute a new activity. The unexpected courses of actions are prompted by a guard condition or an event, and are referred to in SBSD as reaction transitions.

Purpose of the component in the language

The reaction transition components are needed for two reasons. First, in analyzing the actions executed by an entity in a scenario, the reaction indicates to the user that the task performed was not a part of the original mission. In the creation of scenarios it indicates that the transition may, or may not, be taken. Unlike a node that has two or more conditional transitions leaving the node, where the user knows one of the transitions has to be taken, in a reaction transition there is no guarantee that the transition will be taken.

4.2.2.2.4 Temporary Reaction Transition

A temporary reaction transition represents a shift in the activity that an entity is performing to an activity that is not a part of the original set of activities assigned to it.

Syntax

A temporary reaction transition represents the shift in the activity that an entity is performing from the activity represented by the start node to the activity represented by the end node of the transition. The transition is triggered by the condition or event associated with the transition. Upon completion of the activity represented by the end node of the reaction transition, the activity represented by the start node is resumed. A temporary reaction transition is represented by a double line, with arrows pointing to both the start and end task.

Semantics

A temporary reaction transition is used when a modeler wants to specify that in the occurrence of a specific event, the entity should temporarily stop executing the activity represented by the start node of the transition and complete the activity or sequence of activities represented by the end node of the transition. Upon completion of the temporary activity the entity returns to executing the tasks represented by the node represented by the start node of the temporary reaction transition. When the diagram is used to represent events that an entity performed in the simulation, the temporary reaction is used to indicate that the activity at the end of the transition was not a part of the original sequence of assigned activities.

Purpose of component in the language

A temporary reaction is needed because it indicates that the activity that the represented by the start node is not completed before the reaction is taken. Instead the start activity is completed in two parts. Furthermore it reiterates the fact that the activity represented by the end node, may, or may, not take place.

Justification for visual component

The use of an arrow represents that a transition is represented, while the double headed arrow indicates that the transition will eventually return to the activity represented by the start node.

4.2.2.2.5 Permanent Reaction Transition

A permanent reaction represents an action that is permanently taken by an entity and not a part of the original set of assigned tasks.

Syntax

In a permanent reaction transition the entity never returns to complete the task represented by the start node. A permanent reaction transition is represented by a double line, with an arrow pointing to the end task. A condition or guard is associated with a permanent reaction transition to indicate when the transition can be taken.

Semantics

A permanent reaction transition is used when a modeler wants to indicate that on the occurrence of specific events the entity should stop executing the task represented by the start node and proceed to the activity represented by the end node of the transition. Once the entity has started to execute the activity represented by the end node of the simulation it will not return to the start node upon completion of the activity. As with temporary conditional transitions, the transition may or may not be taken.

Purpose of component in the language

If a permanent reaction is taken, it indicates to the user that the activities represented by nodes connected to the start node by regular or conditional transitions are never executed. Therefore the component representing a permanent reaction transition must be different from temporary reaction transition.

4.2.2.11 Examples

Figure 24 shows a SBSD modeling the behavior of an entity in a OneSAF simulation. In the example an USSR Mi - 24 has to wait for the On Order command, then Fly Route for 180 seconds, then Hover 60 seconds before landing. Figure 25 shows an example of a SBSD with a permanent reaction task.

4.2.3 Language Adaptability

Similar to how UML can be extended to be applied to specific domains by creating profiles, SBSD can also be modified to be used by specific simulations. In UML, when the language needs to be refined to apply to a specific domain, a profile for that domain can be created. As discussed in Chapter 2, profiles allow constraints and tag definitions to be applied to the components of UML. Stereotypes specify attributes that

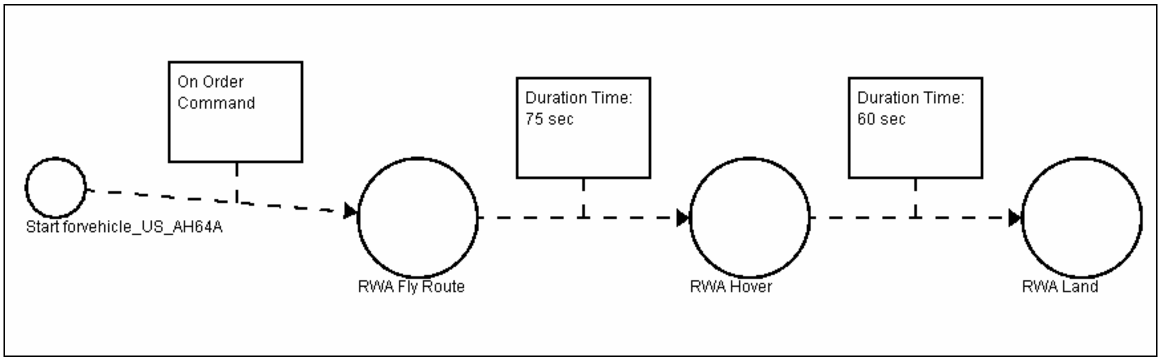


Figure 24. An Example SBSD Diagram

the components of UML diagrams must have and refine the semantics and syntax of the components. Like UML, SBSD is designed to allow similar modifications to be made.

First, the language can syntactically be changed through the addition of constraints on the cardinality of transitions entering and exiting nodes. Currently the only constraint placed on the semantics is that there can be at most one regular transition leaving a node. However to adjust for different simulations, rules on how many of each type of transition can leave or enter a node can be added. For example, in a situation where a sequential path is desired and an entity is not allowed to return to a task once it has been completed (in order to prevent cycles), a constraint that only transitions whose starting nodes cannot be traced back to node A can end in node A could be added to the language.

Another way that the language can be customized is through placing limitations on the type of guards that conditional and reaction transitions can have. For example, in the OneSAF simulation the only type of guards allowed on conditional transitions are a duration constraint, a specification of an HHour, the reception of a message, or reaching a

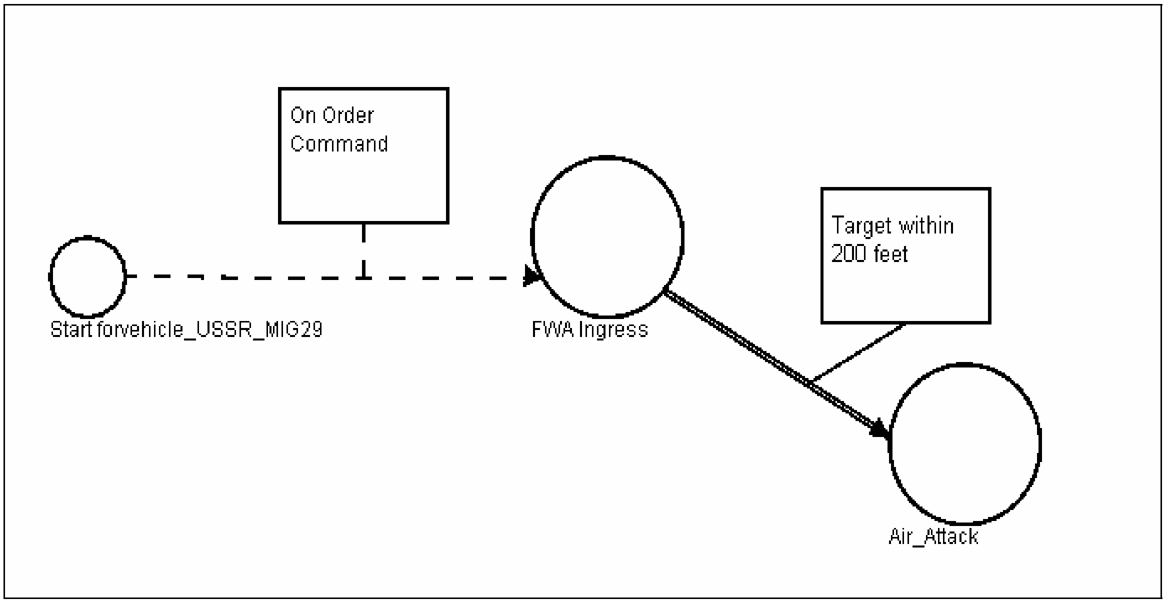


Figure 25. An SBSD Diagram with a Temporary Reaction Transition

control point. Any other type of guard placed in a conditional transition used in a simulation scenario for OneSAF would make the scenario invalid. However, in other simulations there may exist other guards that can be specified.

The language can also be extended through the addition of new components and semantic constraints that are specific toward a simulation. Other types of adaptations require certain attributes to be defined in every node. As long as the change does not affect or contradict the semantics and syntax of the components defined here, the change is valid.

Due to the fact that the adaptations to the language are limited to changes that do not contradict the syntax and semantics of the language, the diagrams for any simulation can be read and processed the same way. The adaptations do not change the basic semantic or syntactical rules of the components and therefore the meanings of the diagrams stay the same. Rather, the adaptations become important when the language is used to build scenarios for specific simulations, because it is these rules that prevent the user from building a syntactically invalid scenario.

4.2.4 Comparison of SBSD to other Behavior Specifications

The nodes in SBSD represent complicated behavior models. It is the intent of the language to allow for the user to modify/view the pre-determined attributes of the activities and not to change the underlying behavioral models they represent. As SBSD was designed to represent the high-level behavior of entities in simulation scenarios and takes into accounts the properties of the domain behavior, SBSD has several advantages over the other behavioral specification models discussed in Chapter 2.

The language is consistent with the representation of high-level behaviors of battlefield entities in simulation, because each component in the language serves a purpose. In the other behavior specification techniques, each technique contained extra components that would make the language inconsistent for the domain of this thesis. For example, ROOMCharts and activity graphs allow for the specification of actions in the transitions or states of the diagrams, and OPNET implemented the Pro-C language. Activity graphs also include branching components for the representation of concurrent actions and activity cycle diagrams contain components used to represent the amount of resources in the process being modeled. By eliminating the extra information and components, the diagrams become easier to understand which decreases the possibility of error when applying the language to the problem domain. Furthermore the removal of the extra components simplifies the language.

SBSD provides for expressiveness in the language as it allows for aspects of the problem domain to be represented that the other diagrams do not. Through reaction transitions it provides a way for reactions to be represented in the diagrams. Activities that are performed as a reaction are interpreted by humans and computers differently than activities that were originally assigned to the entity. Therefore, in order to ensure reactions are interpreted properly, the language representing the behavior in the problem domain needs a way to represent what activities are reactions and what activities are assigned. None of the other specification methods studied provided a way to distinguish the reaction transitions from conditional transitions.

SBSD allows for nodes that represent activity to have attributes and for the attributes of the node to be specified by the user. If the user is not given the ability to specify attributes for activities then a new behavior model would be needed for every possible variation of the behavior. For example, if the attributes of a node could not be modified instead of having an activity that represents one fly route behavior model with route as an attribute, an activity would need to be created to represent a behavioral model for every possible route. Although several of the diagrams allow the specification of parameters activity cycle diagrams and Petri nets do not.

Furthermore, SBSD is capable of representing temporal conditions, which is a necessary component of behavior representation in mission-level simulation scenarios. Through the use of guards on transitions, temporal conditions such as performing a task for three minutes or waiting until a control point is reached to transition to the next activity can be represented through SBSD. Although the logic diagrams used in CBT allow for temporal conditions to be represented in the diagrams through the different type of connectors or transitions, the types of temporal conditions are limited. Similarly, SBSD also allows for there to be no condition on transitions, to indicate that once an activity ends the next activity automatically takes place. ROOMCharts, hierarchical state diagrams, Petri nets, and the state diagrams used in OPNET all require an event to trigger transitions.

SBSD is also a desirable behavior specification diagram compared to the other behavior specification diagrams because it allows for the composability of behaviors, which helps to make the language more scalable, reusable, and composable. Although activity graphs, hierarchical state charts, the state charts used in OPNET, and logic diagrams used in CBT allow for composition of activities, only logic diagrams have a condensed representation of the composed activities. The other diagrams allow for the composition of activities, but the diagram displays the activities inside of the parent activity. SBSD on the other hand, offers a component that represents the composed activities, which can than be expanded or condensed based on the level of detail set by the user. The two representations of the composition of activity help to promote scalability and economy of space in the diagram.

Finally, the diagram has several high-level visual indicators of information not present in the other diagrams. First, the language visually indicates what transitions are guarded. One of the main goals of SBSD is to aid in the understanding of the behavior of the entities through the elements of the diagram. Visual cues, therefore, are important. In the previously discussed diagrams there are no visual indicators for when a trigger for a transition is guarded. For a user to figure out if the transition is guarded they must read the text of the transition. With SBSD the dashed line of the conditional link visually identifies that the transition is guarded, letting the user quickly identify which transitions are conditional. Furthermore, in SBSD the conditions on transitions are easy to identify, while in the ROOMCharts, state diagrams, and activity graphs the user has to locate the trigger in a line of text that may contain other information such as actions in it.

In addition to providing a visual cue to what transactions are guarded, SBSD and the tool supporting the language allow for the visual contraction and expansion of nodes that contain sub-nodes. In ROOMCharts and activity graphs a state can be broken down into sub-states, yet the diagram still shows all the sub-states and transitions between the sub-states. In SBSD less space is taken up by allowing the user to represent a node that is a group of other linked nodes as a variation of a regular node. The node being represented by a double line also quickly indicates to the user that the node can be expanded. Through the tool developed in conjunction with the language the user is given the ability to expand and contract the nodes containing sub-nodes to the desired level of detail.

In Chapter 2, several properties were identified as being necessary to the representation of high-level behavior in simulations. These properties included the ability of the language to represent reactions and temporal conditions. The language should also be composable, have a higher level of abstraction, allow for parameters to be represented in the activity nodes, and have its major focus be on the activities and not the transitions. Table 3 is a comparison of how SBSD compares to other behavior specification techniques in respect to these properties.

Table 3. Comparison of Behavioral Specification Techniques

	Reactions	Parameters	Temporal Conditions	Composability	Focus on Activities	Higher Level of Abstraction
SBSD	Yes	Yes	Yes	Yes	Yes	Yes
ROOMCharts	No	Yes	Yes	No	Yes	Yes
Activity Graphs	No	No	Yes	Yes	Yes	Yes
Activity Cycle Diagrams	No	Yes	No	No	Yes	Yes
State Transition Diagrams	No	No	Yes	Yes	No	No
Petri nets	No	No	Yes	No	No	No
Hierarchical FSM	No	Yes	Yes	Yes	Yes	No
Logic Diagrams	No	Yes	Yes	Yes	Yes	No

4.3 Application of Treemaps

The behavior of the entities is not the only item of interest when looking at simulation scenarios. Another area of interest is the hierarchy between the entities in the scenario. One approach used in information visualization to display hierarchical information is treemaps. Treemaps are a hierarchical interactive visualization method used for presenting hierarchical information. Two important features of treemaps are that they use 100% of the designated display, and that they are interactive. By being interactive, treemaps allow users to specify the structure and content of the hierarchical information displayed in the treemap.

4.3.1 Structure of Treemaps

In a treemap, the tree structure is represented by partitioning the display space into a collection of rectangular bounding boxes. The size of the bounding boxes are determined by the weight of the nodes inside of the box, while the drawing of the nodes inside the bounding box are determined by the content of the nodes and can be interactively controlled.   Properties that the user can have control over include colors, borders, etc.

4.3.2 Treemaps applied to Simulation Scenarios

Treemaps can be applied to simulation scenarios by using the command structure as the determination of what units and entities serve as bounding boxes and what units and entities appear in each bounding box. For example, a US M1 company is composed of two vehicles and three platoons. Each platoon is then composed of four vehicles. Figure 26 shows the company structure discussed above in a standard organization chart, while Figure 27 shows the company represented by a treemap.

In the treemap, the box representing a unit or entity is always in the bounding box that represents the commanding unit of the entity. Furthermore, the sizes of the units and entities in the boxes are proportional to the number of entities and units that the unit or entity is in charge of. Therefore, a vehicle will always be smaller then a platoon when they are in the same bounding box. By adjusting size of the boxes for the size of the units it allows for, units and entities that are larger more space to display information. It also allows the user to look at the treemap and quickly identify which units are the largest. The size of the units in the treemaps can also be changed to represent other factors such as health or ammunition supply of the units. In the organizational chart, it is difficult to tell what size the units are, and the organizational chart has a lot of unused white space.

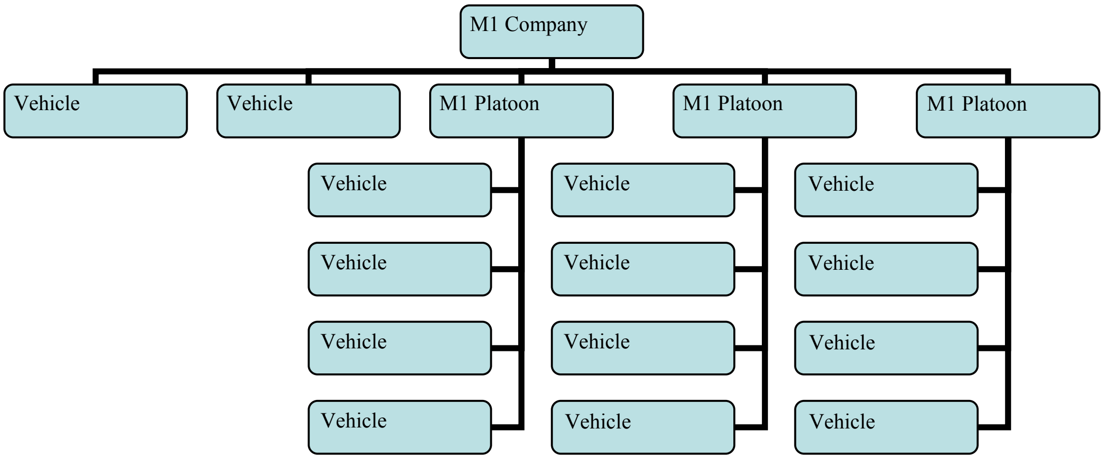


Figure 26. Organizational Chart of an US M1 Company

Furthermore, the organizational chart is not dynamic and does not allow the user to change its properties.

Other properties of the treemap can also be changed to display information about the entities that the user might be interested in. One such property is color. By setting

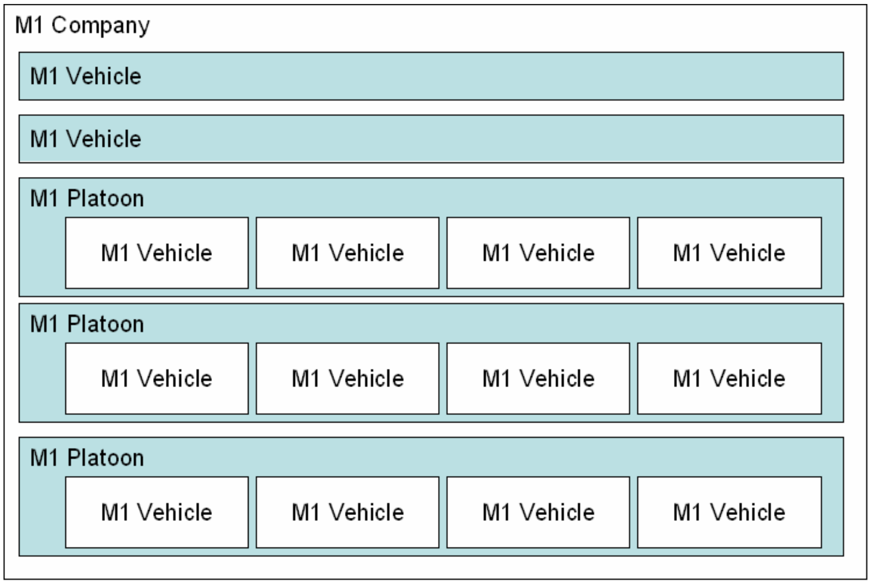


Figure 27. Treemap of an US M1 Company

the color of units to portray certain information such as capability or the assignment of a mission, the treemap allows the user to quickly interpret the chart and get the desired information. Furthermore, as in SBSDs the treemaps can be adapted to meet the needs of specific simulations. The adaptations can define what property determines whether entities are represented by bounding boxes, what property determines the weight of the boxes, and what color the boxes are.

4.4 Summary

SBSDs are used to visually represent the behavior assigned to entities in simulation scenarios. SBSDs are a deviation of process dependency charts and extended finite state diagrams. When compared to the behavior specification techniques discussed in Chapter 2, SBSD proves to be better suited than the other techniques for the representation of activities executed by entities in simulation scenarios. The treemap representation of a hierarchy makes better use of the space allocated for the chart and gives visual indicators as to the size of the entities and units.

V. Implementation and Case Studies

5.1 Introduction

In order to demonstrate the applicability of SBSD and evaluate SBSD against the criteria defined in Chapter 3, the language was applied to simulation scenarios for the One Semi-Automated Forces (OneSAF) simulation. The case studies performed using OneSAF provide a basis to evaluate SBSD. To demonstrate the benefits of applying treemaps to simulation scenarios, treemaps were applied to several OneSAF simulation scenarios.

5.2 Description of OneSAF

OneSAF is a composable simulation currently in test and development by the United States Army. As defined by the history of OneSAF, "OneSAF will be a composable, next generation computer generated forces that can represent a full range of operations, systems, and control process from individual combatant and platform to battalion level, with a variable level of fidelity that supports all modeling and simulation (M&S) domains"[STR02A]. The finished simulation is intended for the training of battalion and brigade commanders and their staff. OneSAF is implemented through the Object Test Baseline (OTB). For this research the OneSAF TestBed Baseline Version 1.0 was used.

5.2.1 Behavior Architecture of OneSAF:

In OneSAF, simulation scenarios are composed of battlefield entities and the behavior assigned to the entities. The behavior of the entities in the simulation is controlled through tasks, task frames, and missions. A task is defined as, "a behavior performed by an OTB SAF entity or unit [STR03d]." A task is defined by its characteristic parameters. The parameters have default values which can be overridden by the user.

The OTB System Design Document defines several properties of tasks. First, tasks can be done as a unit. Therefore, when a task is assigned to an entity in a hierarchy that entity along with its children will perform the assigned task Next, tasks are sequenced together in order to achieve a mission. A sequence of tasks can be grouped together and have an objective. Certain tasks, like monitoring, can also take place continuously regardless of what task is being performed for the mission. Each task is also defined in the system and represents a behavior model. Therefore tasks are representations of actual battlefield behavior and serve as details into the implementation of the system [STR03b].

Tasks are grouped together into task frames, which are groups of tasks that execute concurrently. Task frames typically contain move, shoot, coordinate, and react tasks. Task frames are used to represent a phase in a mission. Task frames can be assigned by the user or by the simulation software. Examples of times when task frames are assigned by simulation software are when high-level tasks (such as those assigned to a battalion) assign tasks to subordinates, or when reactive tasks construct and execute tasks frames in response to changes in the battlefield [STR03B].

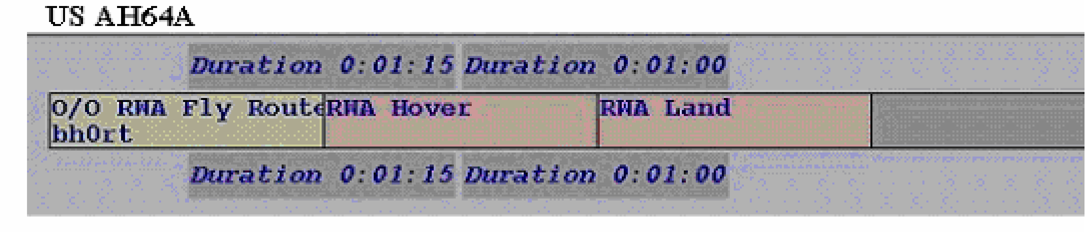
Task frames are linked together in a sequence to form missions. Enabling tasks can be placed between the task frames and serve as a way to represent predicted contingencies. Enabling tasks serve as predicate functions, and a task that has an enabling task can only be executed after the enabling task evaluates to true.

All the task frames assigned to a unit are stored in a task frame stack. In a task frame stack the topmost task frame is active, while the rest of the task frames are suspended. New task frames can be added to the top of the stack as a result of a reactive task or by the user observing the execution of the scenario [STR03b].

5.2.2 User Controllable Behavior in OneSAF:

In OneSAF the user can control behavior by creating pre-planned missions, setting up reactions, and issuing immediate commands. Users set up pre-planned missions through execution matrices. Figure 28 shows an example of an execution matrix used in the OTB. Execution matrices in the OTB are based on the execution matrixes used by the Army. The execution matrix divides the battle into phases, and shows what each unit should be doing at each phase. In setting up a mission, users assign a task frame to units or entities in the simulation using the execution matrix. When they assign the task frame to the unit, the user is given a set of parameters concerning the task frame that they can adjust. Location, formation type, and maximum speed are all examples of parameters the user has control over [STR03c].

In between the task frames users can also place enabling tasks or conditions that have to be met before the execution of that task frame can begin. In OTB Version 1.0 a duration, HHour, control point, or message from another entity are valid enabling tasks that the user is able to select.



US AH64A

Duration 0:01:15 Duration 0:01:00

O/O RWA Fly Route bh0rt | RWA Hover | RWA Land

Duration 0:01:15 Duration 0:01:00

Figure 28. Execution Matrix Used in the OTB

Reactive triggers, which are represented by a task, are used to implement reactions. Reactive triggers monitor the current conditions of the battlefield and then invoke the corresponding task when the conditions of the task are met. Users can define situations and map reactions to them through the parameters on reactions. Users also have the ability to modify an existing task or assign new tasks to units and entities during the execution of the simulation. Users can also interrupt the current mission to perform new tasks, and then have the entity return to executing the assigned mission [STR03C].

5.3 Application of SBSD to OneSAF Simulation Scenarios

In OneSAF the behavior that the SBSD is intended to represent is the high-level behavior represented by task frames. The users of the simulation are not concerned with the tasks, nor the detailed behavioral models behind the tasks. Instead, they are interested in the task frame they are assigning and the parameters that they can adjust. In the case studies presented at the end of this chapter it shown that the language is able to accurately represent the set of task frames and the properties of the task frames assigned to units and entities in OneSAF.

5.3.1 Simulation Scenarios and Persistent Objects in OneSAF

In OTB Version 1 simulation scenarios, all the persistent objects in the simulation are saved in a text file. The use of a text file allows for scenarios from older versions of the simulation to be re-used in newer versions and for manual changing of the simulation scenario through a text editor. The use of a text file also allows other programs, such as the one written for this research access to information stored in the scenario.

Included in the persistent objects written out to the text file are the entities and units in the scenario and the task frames that have been assigned to or completed by them. The persistent objects store the parameters assigned by the user for the objects and tasks assigned to the entities along with other information. Figure 29 gives an example of a text version of a task frame object in a scenario file. Although the text file of the scenario is readable by humans, it is tedious to interpret. The references used in the objects are through other object names, and putting together sequences of task frames, or finding out what task frames belong to what task is a time consuming process. Furthermore, the text files are not small and a scenario representing four vehicles and their missions is 241 pages long.

5.4 Program Supporting SBSD and OneSAF Simulations

In order to apply SBSD to OneSAF, a Java program Simulation Scenario Specification Tool (SSST) was developed to read the scenario files and allow the user to view, edit, and assign behavior to entities and units in the scenario. SSST is divided into two parts. The first part reads in the simulation scenario and then recreates the structure of the scenario in memory. A graphical user interface (GUI) makes up the second part of the program and it is through the GUI that the users can view, modify, and create behavior for the entities in the scenario. A third part of SSST needs to be implemented in order to write out updated simulation scenarios, but is left for future work.

```
("objectClassTaskFrame" "object7"
        (
           ("name" "FWA Return to Base")
           ("opaque" "true")
           ("destroyWhenDone" "false")
           ("assigned" "true")
           ("instruction" "TIIPopOpaque")
           ("unit" "object9")
           ("commandingUnit" "no object")
           ("nextStackFrame" "no object")
           ("previousMissionFrame" "object10")
           ("sponsoringTask" "no object")
           ("logicStack"
              (
                 (128)
                 (0 1 253 255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                    0 0 0 0 0 0 0 0 0)
              )
           )
           ("etaskCount" 2)
           ("enablingTask"
              (
                 (2)
                 ("object11" "object0")
              )
           )
        )
     )
```

Figure 29. Representation of aTask Frame in a Scenario File

SSST was implemented in Java for three reasons. First Java, is portable and can be run on several platforms. Second, Java has a several libraries and tools that support GUI development. Finally, since Java focuses on objects, the concept of entities and tasks assigned to the entities maps into Java without a lot of complexity.

5.4.1 Parsing the Simulation Scenario Files

The first part of SSST is designed to read in the scenario files and then reconstruct the model of the unit hierarchy of the entities in the scenario and the missions assigned to each entity respectively. The simulation scenarios files created by OneSAF consist of a list of objects. The objects represent the entities in the simulation, the state of the entities, the task frames assigned to the entities, the tasks that make up the task frames, and the state of the tasks. Each object contains an object ID, an object type, and a list of attributes and values. All references in the simulation file are made by using the object ID of the object being referenced.

The storage of the entities and task frames assigned to the entities is handled through the unit class, taskframe class, tasks class, enabling task class, and scenario class. The unit class holds information about each entity in the system, along with references to its commanding unit, subordinates, the first task frame in the mission, and the task frame the entity was executing or waiting to execute when the scenario was saved. The task frame class holds references to the unit a task frame is assigned to, to the next and previous task frame in the mission and any enabling tasks associated with the task. The scenario class holds the list of the battlefield entities and units in the scenario.

In order to set up the information needed for the scenario development two passes of the simulation scenario are made. In the first pass, all the task frames in the scenario are read in and stored in the taskframe class. Then in the second pass all the units and entities in the simulation are read in.

The program then calls methods that set up references between the units and the task frames, and between the task frames. The references between the task frames work like a linked list with each task frame containing a reference to the previous task frame and to the next task frame in the mission. In the case where a task was a reaction, a reference to the task frame the reaction was called from is also stored. By storing this reference it allows for reaction tasks to be displayed and referenced. References to enabling tasks, or tasks that control when the task frame can be executed are also set up at that time. The enabling task class is an abstract class, which is extended by concrete classes representing the conditions allowed in OTB Version 1.0. Figure 30 shows how task frames and entities are linked in the program. After the objects in the scenario have been loaded into the Java application and the references set up, the scenario is ready to be viewed and modified through the GUI.

5.4.2 The Graphical User Interface

The second part of the Java program is the GUI. It is through the GUI that users are able to view tasks, modify existing tasks, and create new tasks. Figure 31 is a screen shot of the GUI. The GUI can be broken up into four different parts. Area 1 is a tool bar. On the toolbar are buttons which represent the different components in the SBSD language along with a selection and clear button. It works like the tool bar in most

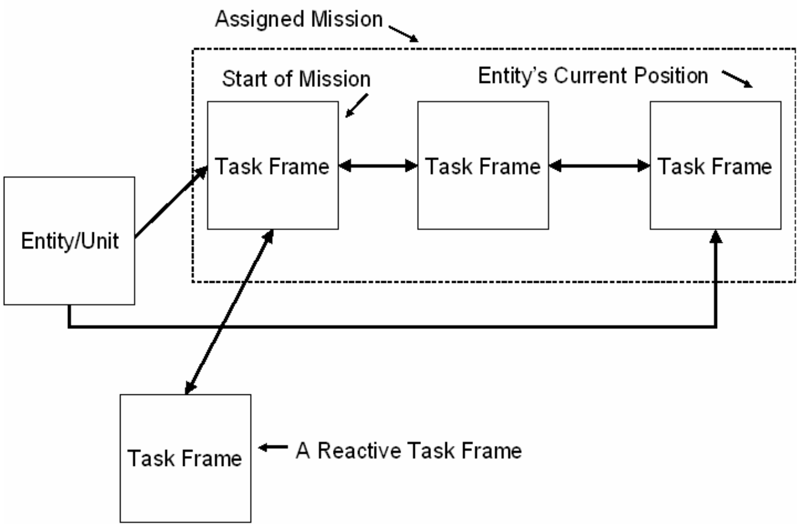


Figure 30. References Between Task Frames and Units in the Java Program

programs. To add a component onto the screen the user simply selects the component they want in the tool box and then clicks the mouse in area 4 where the component should be placed. Area 2 shows the available tasks and missions that the user can click on. By selecting a task frame the tasks performed in the task frame appear below the missions. Area 3 displays the units and entities involved in the scenario. By clicking on the entities, the assigned or completed behavior of the entity is displayed in Area 4. By right clicking the user is given the option of editing the behavior or viewing the text of the scenario file. Area 4 serves as the canvas for viewing or editing behaviors assigned to the units. The user can also create multi-task nodes through another editor, and then assign the multi-task nodes as part of the mission to the entities in the simulation scenario.

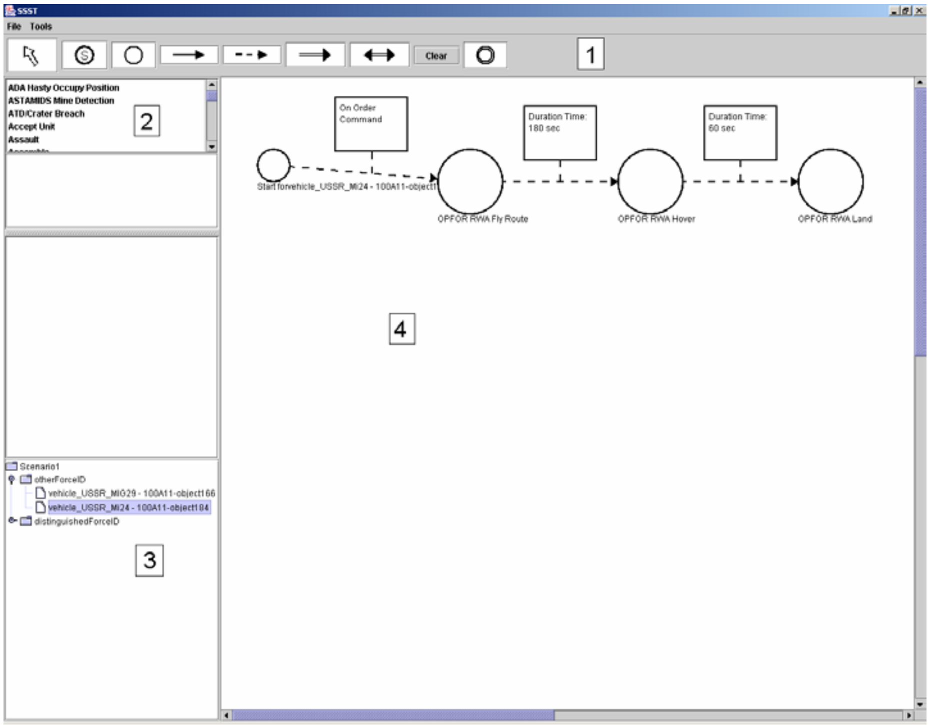


Figure 31. Screen Shot of the Graphical User Interface for SSST

When a multi-task node is inserted into the diagram it can be viewed in the condensed form or in the expanded form. In either form the user is still able to edit the parameters of the individual tasks in the multi-task mission. By allowing users to group commonly used sequences of tasks together it saves time for the user, and takes up less space in the diagram. Figure 32 shows a sequence of task frames with a multi-task node condensed and then expanded.

It is mainly through the creation of multi-task nodes that SSST supports composability. In SSST, the user can create a sequence of atomic nodes and make a new node out of them. That new node can then be used in another sequence of nodes used to create a multi-task node. However to the user, the multi-task node can be treated in the same way as an atomic node. Therefore, complex sequences of tasks can be made up of

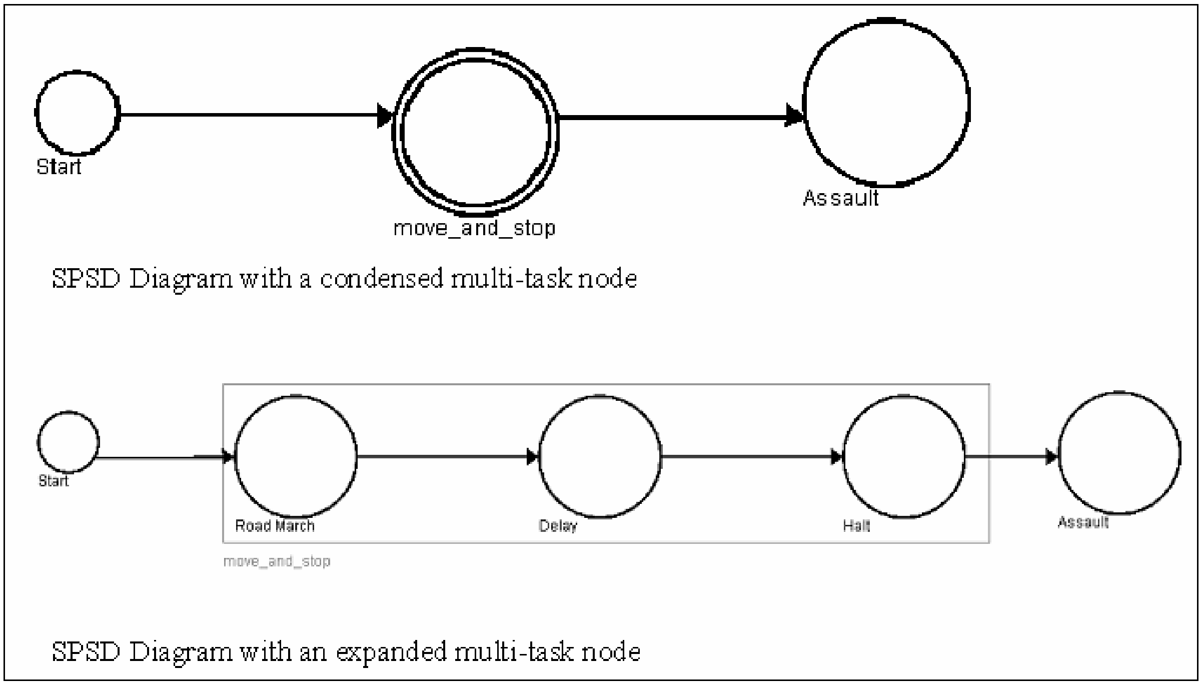

Start
move_and_stop
Assault
SPSD Diagram with a condensed multi-task node
Start
Road March
Delay
Halt
Assault
move_and_stop
SPSD Diagram with an expanded multi-task node

Figure 32. A Mission with a Multi-task Node Condensed and Expanded

sequences of tasks which themselves are composed of sequences of tasks. SSST also supports composability because it lets the user treat the behavior an entity is assigned as a separate component of the entity.

5.4.2.1 Viewing of the Simulation Scenarios

In order to make the GUI adaptable to displaying the scenarios of simulations other than OneSAF, the part of the GUI that displays the behaviors of the entities in the simulation was designed to use the properties of the behaviors for a simulation scenario along with a generic behavior object. In the program, a class called “Behavior Object” is used to represent the information to be displayed in the SBSD diagram. A behavior object contains the name of the behavior, any enabling conditions, and has the potential to list any attributes of a behavior modifiable by the user.

The behavior object can be modified as needed to meet the needs of different simulation scenarios. When the user selects a scenario to load into the GUI, the program parses the scenario into memory. The scenario object then provides the GUI with a list of behavior objects. The GUI then goes through and displays the SBSDs based on the information stored in the behavior objects. Through the use of behavior objects, the parsing of the scenario can be recreated for scenarios of other simulations and as long as the scenario can be parsed into behavior objects, the GUI can easily be adapted to display other simulation scenarios.

5.5 Program Supporting TreeMaps

Another aspect of the SSST is the ability of the program to display the hierarchy of the units in a treemap. Figure 33 shows a screen shot of the GUI developed as part of SSST to apply treemaps to the simulation scenarios. The screen shot shows how the treemap breaks up the hierarchies and places each unit in the bounding box of its commanding unit.

5.5.1 Implemented Capabilities of the TreeMap Program

The purpose of the treemap is to allow the user to view information about the units and entities in the scenario in a manner that facilitates the decision-making process. This is done by using a format that maximizes 100% of the display area and allows user-interaction with the treemap. The treemap also sets the size of each unit and entity relative to that unit or entity's size in the simulation, giving the user a visual indication of the size of each force and unit.

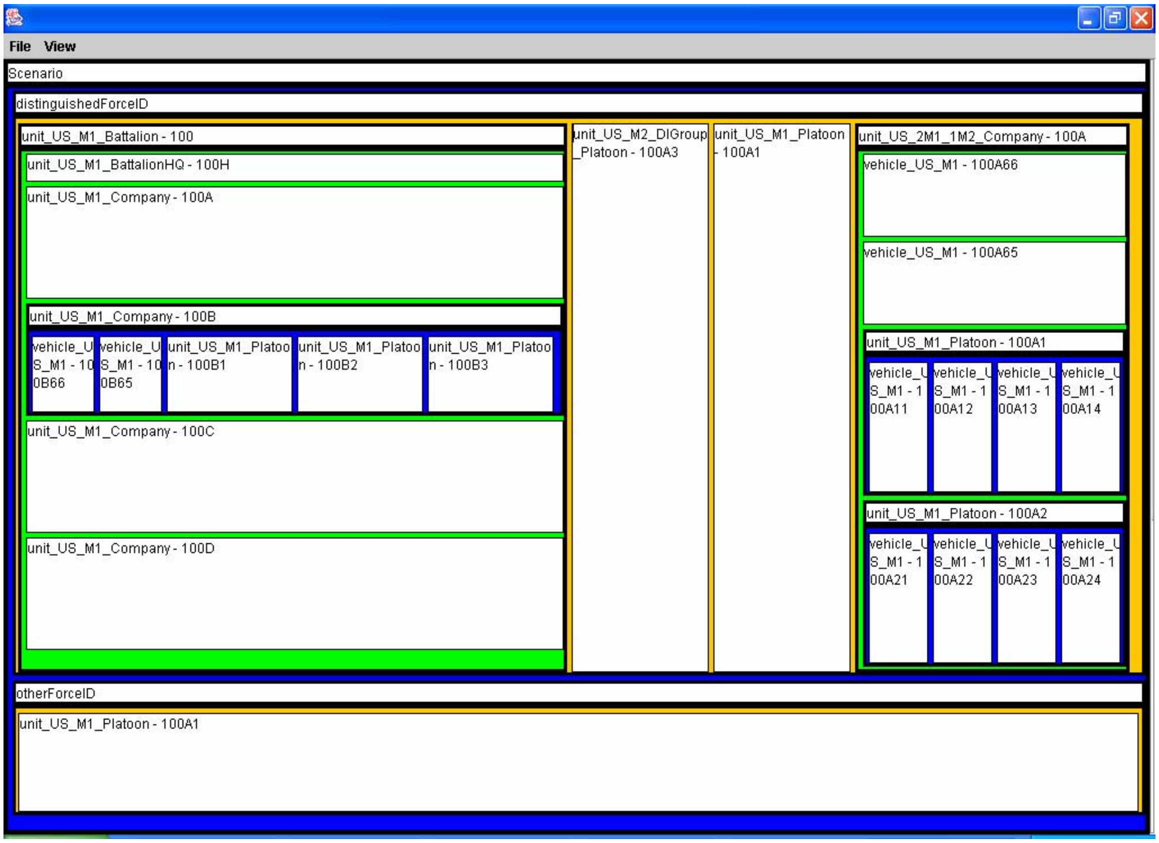

File View
Scenario
distinguishedForceID
unit_US_M1_Battalion - 100
unit_US_M1_BattalionHQ - 100H
unit_US_M1_Company - 100A
unit_US_M1_Company - 100B
vehicle_US_M1 - 100B66
vehicle_US_M1 - 100B65
unit_US_M1_Platoon - 100B1
unit_US_M1_Platoon - 100B2
unit_US_M1_Platoon - 100B3
unit_US_M1_Company - 100C
unit_US_M1_Company - 100D
unit_US_M2_DIGroup_Platoon - 100A3
unit_US_M1_Platoon - 100A1
unit_US_2M1_1M2_Company - 100A
vehicle_US_M1 - 100A66
vehicle_US_M1 - 100A65
unit_US_M1_Platoon - 100A1
vehicle_US_M1 - 100A11
vehicle_US_M1 - 100A12
vehicle_US_M1 - 100A13
vehicle_US_M1 - 100A14
unit_US_M1_Platoon - 100A2
vehicle_US_M1 - 100A21
vehicle_US_M1 - 100A22
vehicle_US_M1 - 100A23
vehicle_US_M1 - 100A24
otherForceID
unit_US_M1_Platoon - 100A1

Figure 33. A Treemap of a OneSAF Scenarios

One way that the user can interact with the treemap is by selecting the level of detail displayed for each unit and entity. When a user clicks on a unit that has subordinates, the subordinate units and entities are displayed within the space allocated to the unit clicked. By right clicking on the name of the unit, the user is given the option to remove the subordinate units.

A second kind of interaction that the user has is the capability of bringing up a unit and entities in a separate frame. By bringing the unit up in a separate frame, the user can get more information about the unit. Information that can be viewed about each unit or entity when selected include a SBSD of the mission assigned to the unit and the attributes of the unit.

A third kind of interaction the user has with the treemap is to change the color of the units and entities in the treemap display information about the units. SSST allows the user to view the capability of each unit. Figure 34, shows a screen shot of the program where the color of the units and entities are determined by the capability of each entity and unit. By using color to represent the capability of the units and entities in the simulation the user can quickly find out which units are the strongest and weakest units. For example if gray is used to represent units with a capability equal to 5 and black is used to represent a unit with a capability of 2, one can determine from the color that a gray unit has more capability then a black unit, After a scenario has been executed this feature also allows the user to quickly determine which units lost the most capability during the scenario execution.

5.6 Case Studies

In order to validate the usefulness of SBSDs, the language was applied to several OneSAF simulation scenarios. The first simulation scenario is an example scenario from the user manual for OneSAF. The rest of the simulation scenarios tested are scenarios provided by the OneSAF Program Office.

5.6.1 Case Study One

The first case study conducted represents the behaviors of entities in a simple simulation scenario. The scenario is taken from Volume One of the user's guide for OTB Version 1.0. It consists of four entities, a USSR MIG-29, a USSR MI-24, a US AH-64A, and a US F-14D. The USSR entities are placed on the "Other Force" side, while the United States aircraft are placed on the "Distinguished Force" side of the battlefield.

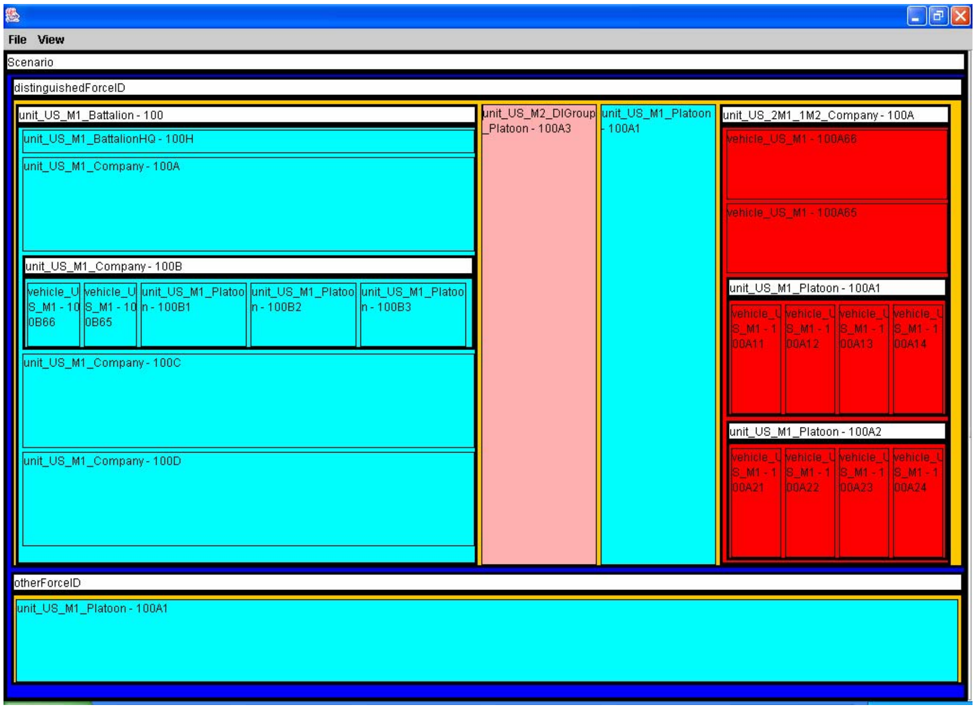

File View
Scenario
distinguishedForceID
unit_US_M1_Battalion - 100
unit_US_M1_BattalionHQ - 100H
unit_US_M1_Company - 100A
unit_US_M1_Company - 100B
vehicle_US_M1 - 100B66
vehicle_US_M1 - 100B65
unit_US_M1_Platoon - 100B1
unit_US_M1_Platoon - 100B2
unit_US_M1_Platoon - 100B3
unit_US_M1_Company - 100C
unit_US_M1_Company - 100D
unit_US_M2_DIGroup_Platoon - 100A3
unit_US_M1_Platoon - 100A1
unit_US_2M1_1M2_Company - 100A
vehicle_US_M1 - 100A66
vehicle_US_M1 - 100A65
unit_US_M1_Platoon - 100A1
vehicle_US_M1 - 100A11
vehicle_US_M1 - 100A12
vehicle_US_M1 - 100A13
vehicle_US_M1 - 100A14
unit_US_M1_Platoon - 100A2
vehicle_US_M1 - 100A21
vehicle_US_M1 - 100A22
vehicle_US_M1 - 100A23
vehicle_US_M1 - 100A24
otherForceID
unit_US_M1_Platoon - 100A1

Figure 34. A Treemap Coloring the Units Based on Capability

Each of the entities in the scenario is assigned a short mission to complete by the developer of the scenario. Figure 35 shows the execution matrixes for each of the entities in the scenario.

In order to evaluate the scenario, a file representing the state of the scenario before it was executed and a file representing the state of the scenario after it was executed were made. Figure 36 shows the SBSD for the state of scenario before the scenario was executed and 37 show SBSD of the scenario after the scenario was executed.

From the second diagram one can see that the USSR MIG-29 took the permanent reaction transition to execute the task air attack, and did not complete the ingress

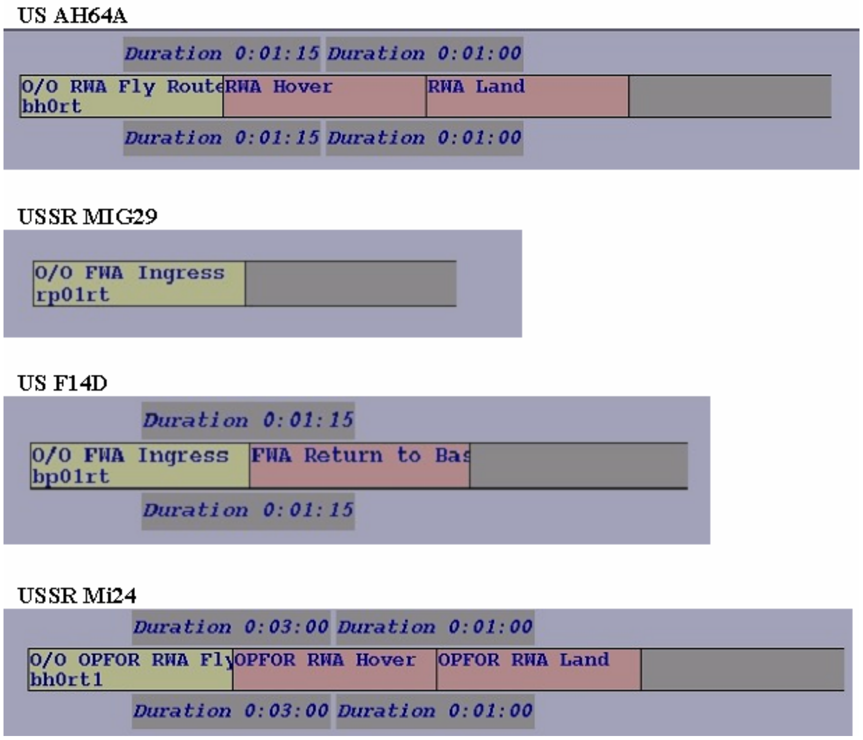


Figure 35. Execution Matrices for the Entities in Case Study One

operation. In the execution matrix, there is no way to note that this action was taken, or that the action that was assigned before the attack took place.

What the diagram does not show is the air attack reaction tasks frames that the US F-14D entity executed in response to sighting the USSR MIG-29 and the USSR MI-24. However, these missing pieces are not due to deficiencies in the language, but occur since insufficient information to reconstruct them exists in the scenario files. In the current version of the object test baseline used to run the scenarios, once a reaction task is completed the simulation deletes the task frame representing the task from the database used to generate the simulation scenario files. Tasks that are assigned to the entity by the user, on the other hand, are saved even after execution. With modifications of the OTB

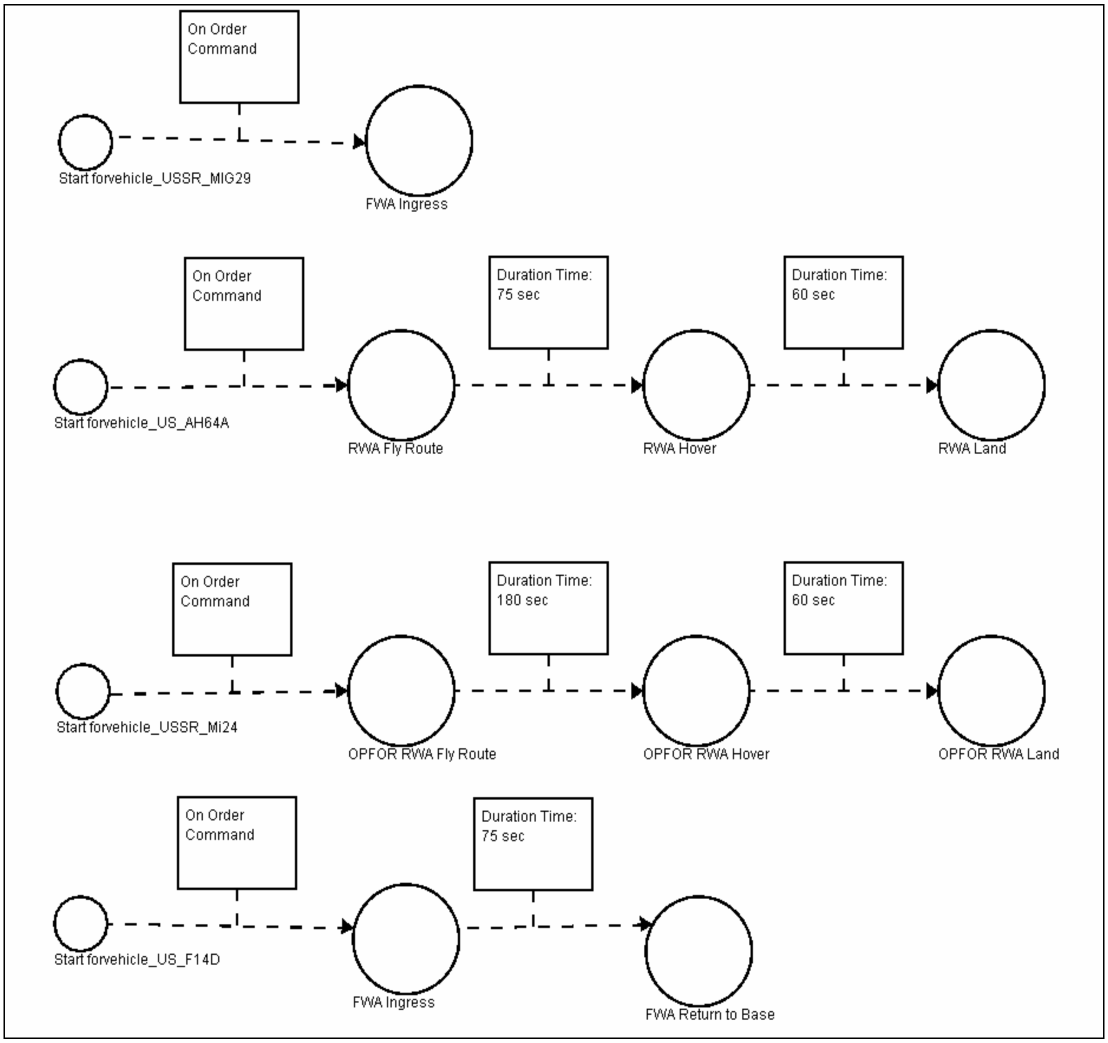


Figure 36. The SBSD Diagram Showing the Behaviors Assigned to the Entities

that allow for the entire sequence of task frames executed by an entity to be passed to the Java program, the tool should be able to correctly represent the behaviors of the entities. Figure 38 shows the diagram with the reaction behaviors performed by the US F-14D. The diagram can further be expanded to include the attributes of the behaviors such as the location of the base the entity is returning to and the target of each of the Air Attack behaviors.

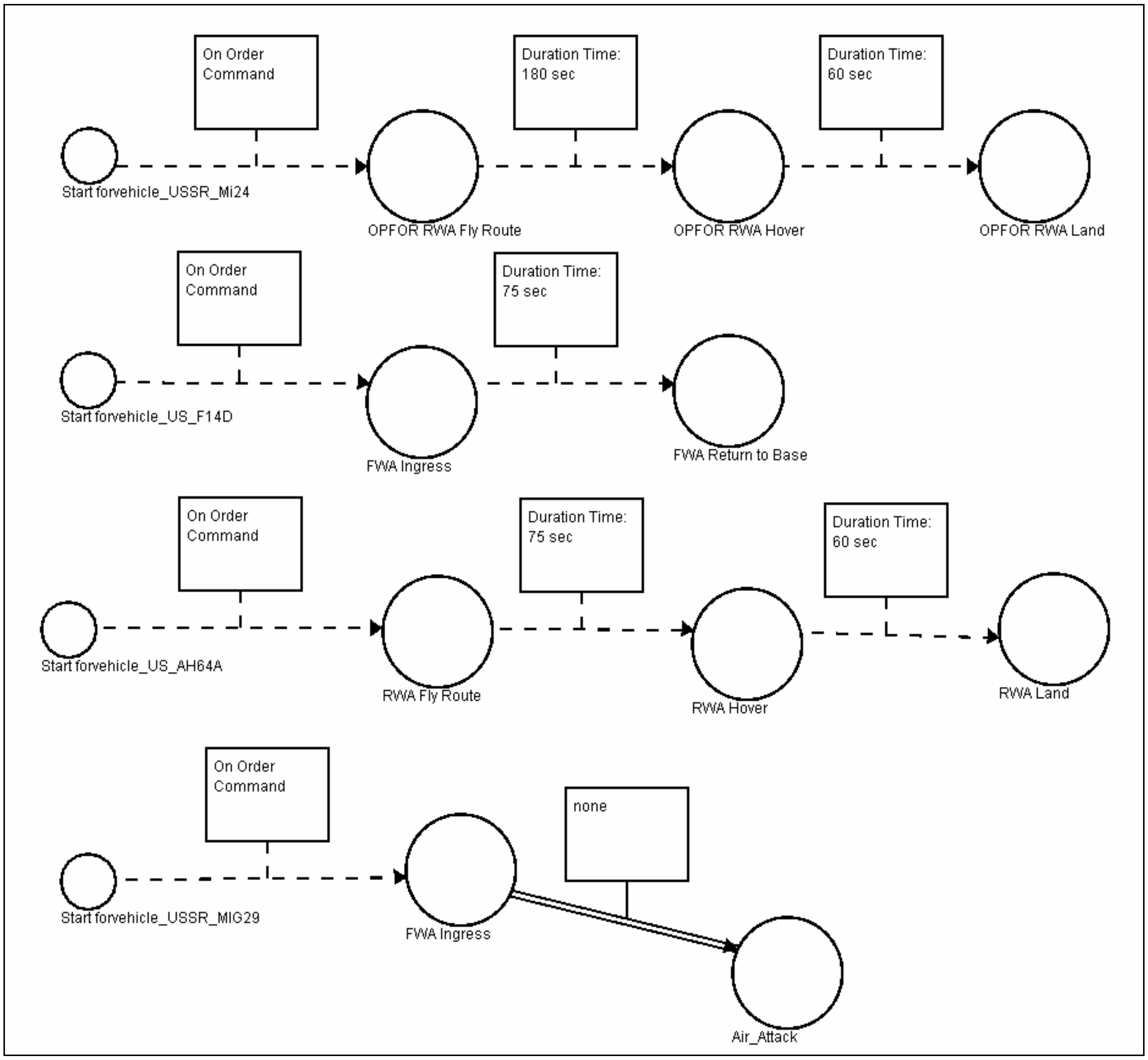


Figure 37. The SBSD Diagrams for the Behaviors Actually Performed by the Entities in Case Study One

5.6.2 Case Study Two

The second case study represents the behaviors of entities in a scenario that is slightly more complex scenario than the scenario used in the first case study. The scenario used in case study two is from the OneSAF program office. The scenario is

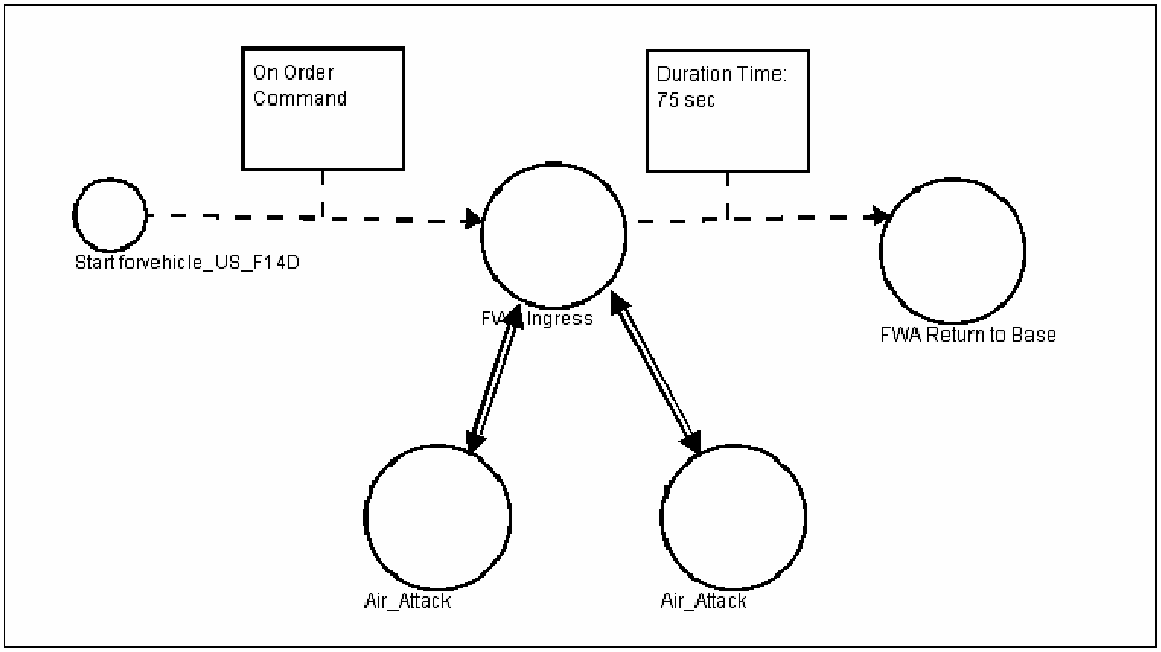


Figure 38. Actions Performed by US F14D, including the Reaction Tasks

made up of two forces. The "Other Force" is made up of four USSR MIG-27D aircraft and four USSR T80 platoons. Each T80 platoon consists of three T80 ground vehicles. The "Distinguished Force" is made up of four US F-16 aircraft, two US A-10 aircraft, and three US M1A platoons. Each M1A platoon consists of four M1A ground vehicles.

Unlike the previous scenarios, where each entity or unit in the simulation was assigned a sequence of multiple task frames to execute, each of the entities and units in this simulation was only assigned one task frame. All of the aircraft for both sides were assigned the attack ground target task frame. Two of the USSR T80 platoons were assigned the OPFOR road march task frame, and the remaining USSR T80 platoon was assigned a traveling overwatch task frame. The US platoons were all assigned the assault task frame. Because each of the entities were only assigned one task frame, the only benefit of SBSD diagrams over execution matrices is graphical visualization. Figure 39 shows one of the SBSD diagrams generated in the second case study.

From the SBSD diagrams generated it looks like there is a low level of activity in the scenario. However, when the scenario is executed all of the aircraft shoot targets, and two of the ground platoons perform reaction tasks based on the detection of enemy aircraft. Therefore, it is evident that some the task frames consist of more then one activity.

In order to deal with the complexity of certain task frames SSST was expanded to give the user the ability to define SBSDs for task frames. Due to the high level of complexity of the behavior represented by some of the task frames and in order to stay at a higher level of abstraction the SBSDs for task frames are created through human input, instead of by parsing a task frame file. The diagrams created are based on the task frame and task descriptions in the User Manual for OTB Version One. In this instance, the SBSDs are only a representation of the behavior contained in the task frame, and cannot be converted into code for use by the OTB. Figures 40, 41, and 42 show the SBSDs for the assigned task frames. The OPFOR March task frame is not defined in the User Manuel for the OTB.

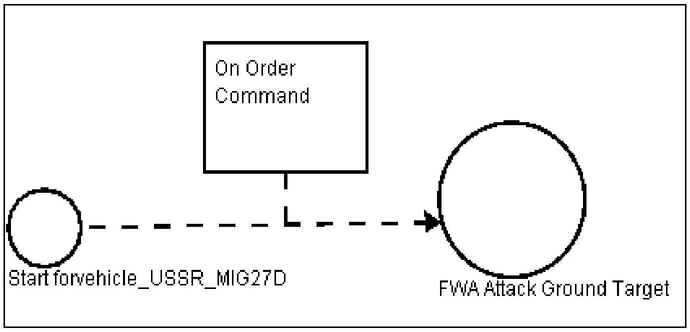


Figure 39. An SBSD Diagram for a USSR MIG27D in Case Study Two

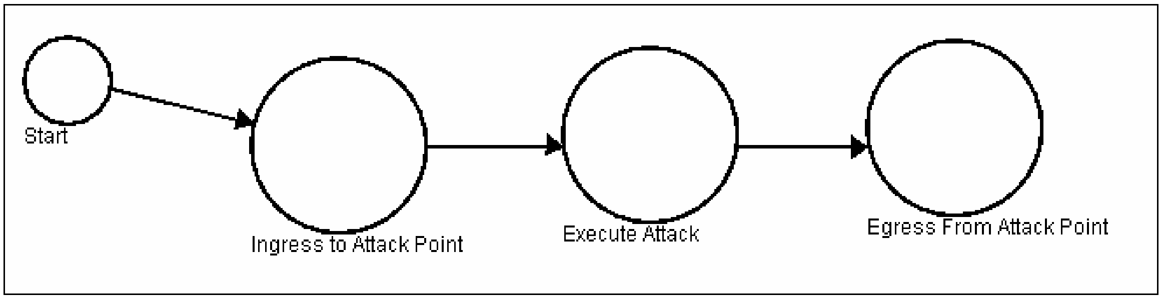


Figure 40. An SBSD Diagram of an Attack Ground Target Task Frame

The SBSDs of the task frames are a much better representation of the behavior that the entity is performing than the representation provided by an atomic node. From the atomic node representation there is no information about the possible reactions that might take place, or the break down of sub-activities that the task frame performs. For example, in the SBSD of the ground target frame one can see that there are three different

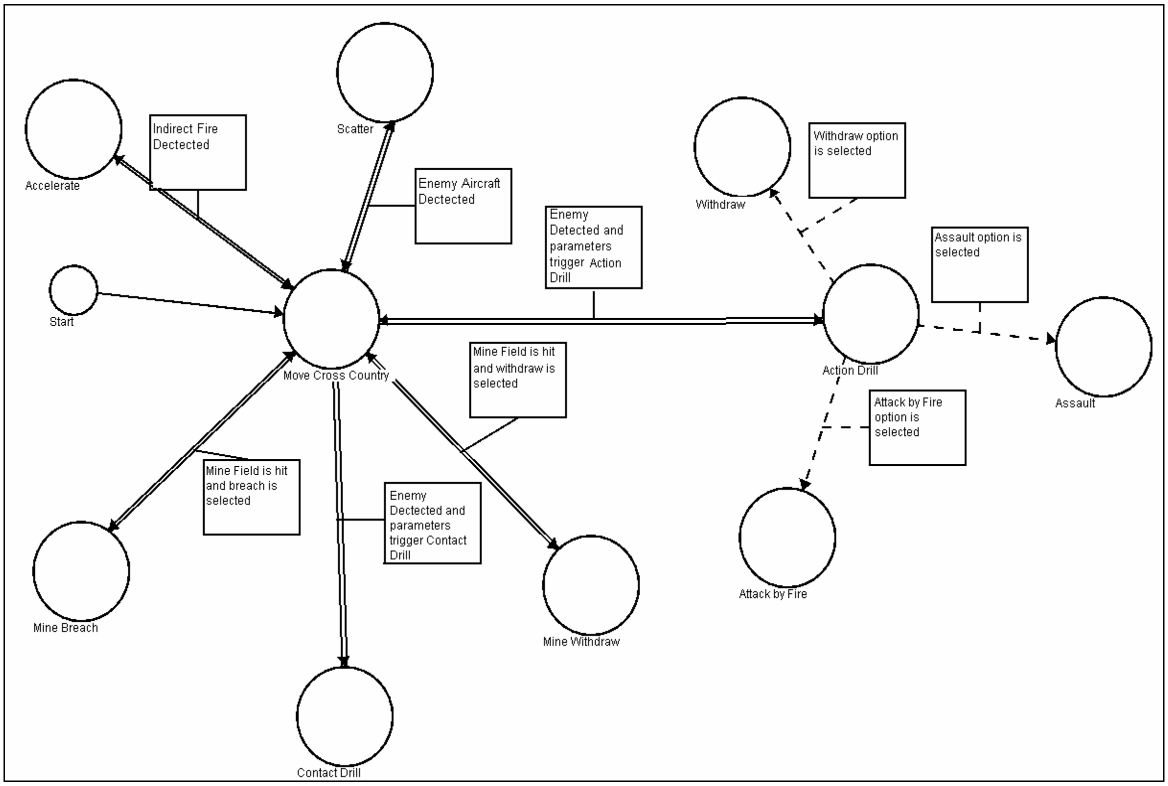

Accelerate
Indirect Fire Dectected
Scatter
Enemy Aircraft Dectected
Enemy Dectected and parameters trigger Action Drill
Withdraw
Withdraw option is selected
Assault option is selected
Start
Move Cross Country
Action Drill
Assault
Mine Field is hit and withdraw is selected
Attack by Fire option is selected
Mine Field is hit and breach is selected
Enemy Dectected and parameters trigger Contact Drill
Mine Breach
Mine Withdraw
Attack by Fire
Contact Drill

Figure 41. An SBSD Diagram of the Traveling Overwatch Task Frame

parts to the task frame. The SBSD diagram for the assault task frame (Figure 42) also indicates that in the event an enemy air vehicle is detected that the ground unit should scatter.

At this point there are two ways the SBSDs of task frames can be integrated into SSST. The first is to allow the task frames to be represented as atomic nodes, and make users right click on them to view the SBSD diagram of them. This option is allowed because the user would not have the ability to change the SBSD for the task frames. Therefore the SBSD for the task frames represent behavior of the entity at a lower level of abstraciton and the task frames would still be the lowest level of detail available to a person creating a scenario. The second option is to allow task frames that have SBSD diagrams associated with them to be represented by multi-task nodes. By using multi-task nodes to represent the tasks, the user is given control over the level of detail that is shown in the diagram. The risk of representing the task frames as multi-task nodes is that they could get confused with multi-task nodes that are changeable by the user. This issue is further addressed in Case Study Three.

5.6.3 Case Study Three

The third case study looks at a scenario that involves the pickup of three US IC Fire Teams and is from the OneSAF Program Office. The scenario is composed of three US IC Fire Team A units, one US CH-47D flight of 3, and one US Fire AH-64D flight of 2. Figure 43 shows the execution matrices for the entities, while Figure 44 shows the SBSD Diagrams created by SSST for the execution matrices in Figure 43.

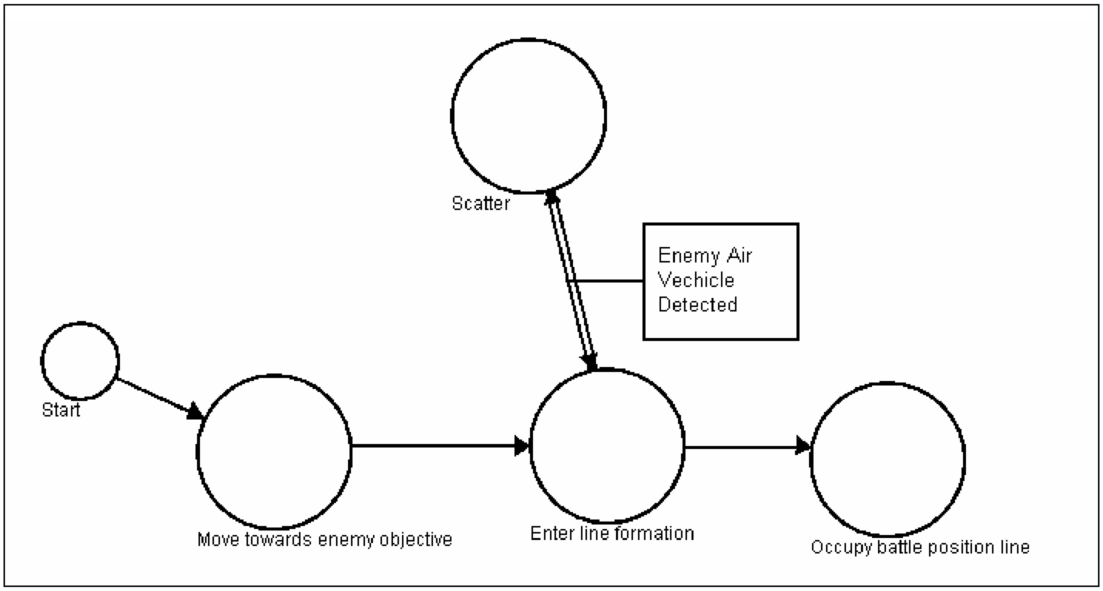
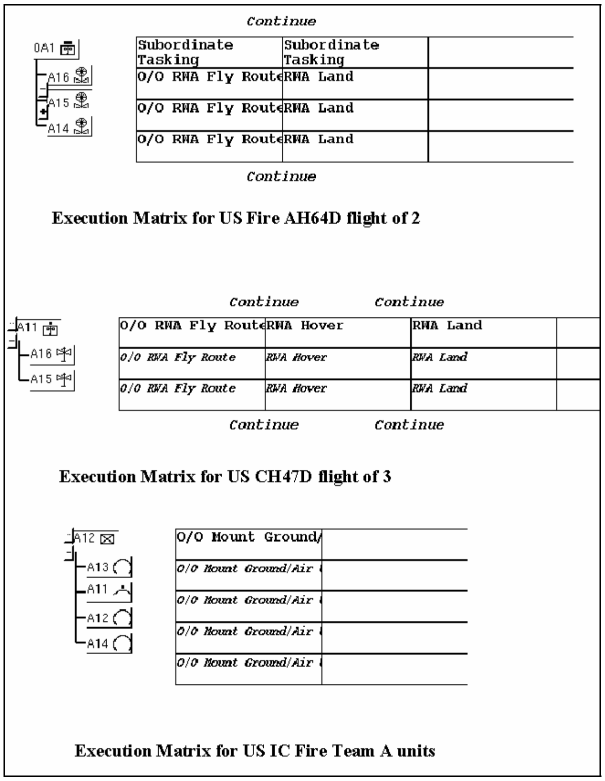


Figure 42. An SBSD Diagram of an Assault Task Frame

The behavior of the entities in case study three are similar to the behaviors of the entities in case study one and case study two. The behaviors assigned to the entities are similar to the behaviors assigned to the entities in case study one, in that with the exception of the mount task frame assigned to the Fire Team A units, all the units are assigned multiple tasks frames. However, with the exception of the Mount Ground Air/Unit each of the tasks frames assigned have complex behavior that can be represented by a SBSD. The complex behavior of the task frames is similar to the behavior of the task frames assigned in scenario two.

Figure 45 shows the SBSD for the generic RWA Fly Route task frame. In the SBSD for the RWA fly route task frame, the react to ground contact activity takes up a lot of space on the diagram, and is composed of several sub-activities. Therefore, it becomes a candidate to be converted into a multi-task node. Figure 46 shows the react to ground contact sub-activity expressed as an expanded multi-task node.



Continue

Subordinate Tasking	Subordinate Tasking	
O/O RWA Fly Route	RWA Land	
O/O RWA Fly Route	RWA Land	
O/O RWA Fly Route	RWA Land	

Continue

Execution Matrix for US Fire AH64D flight of 2

Continue *Continue*

O/O RWA Fly Route	RWA Hover	RWA Land	
O/O RWA Fly Route	*RWA Hover*	*RWA Land*	
O/O RWA Fly Route	*RWA Hover*	*RWA Land*	

Continue *Continue*

Execution Matrix for US CH47D flight of 3

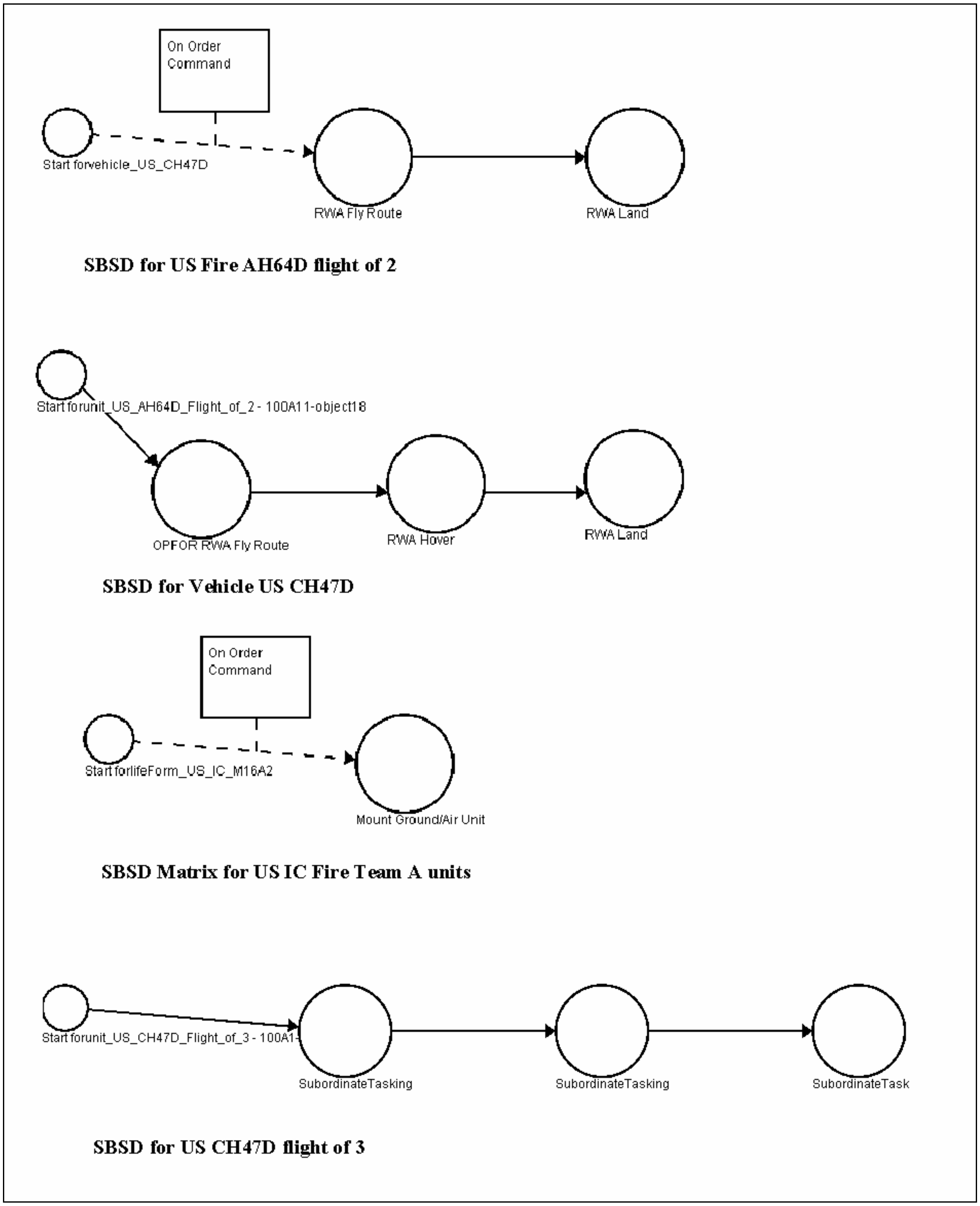
O/O Mount Ground/	
O/O Mount Ground/Air	
O/O Mount Ground/Air	
O/O Mount Ground/Air	
O/O Mount Ground/Air	

Execution Matrix for US IC Fire Team A units

Figure 43. Execution Matrices for Units in Case Study Three

The RWA Hover and RWA Land task frames also contain the react to ground contact sub-activity. As a result of the sub-activity being represented by a multi-task node and not an atomic node it can be condensed in the SBSDs for the RWA Hover and RWA Land task. Figure 47 shows the SBSD for the Hover Task Frame representing the React to Ground Contact sub activity as a multi-task node. The condensed representation makes the SBSD easier to read, but also give the user the ability to change the level of detail shown in the diagram.

For certain task frames the SBSD diagrams of task frames for an entity in an already created scenario may be different then the SBSD diagrams created for taskframe that has not been assigned to an entity. The difference occurs when a user has to choose



SBSD for US Fire AH64D flight of 2

SBSD for Vehicle US CH47D

SBSD Matrix for US IC Fire Team A units

SBSD for US CH47D flight of 3

Figure 44. SBSD Diagrams for the Execution Matrices in Figure 43.

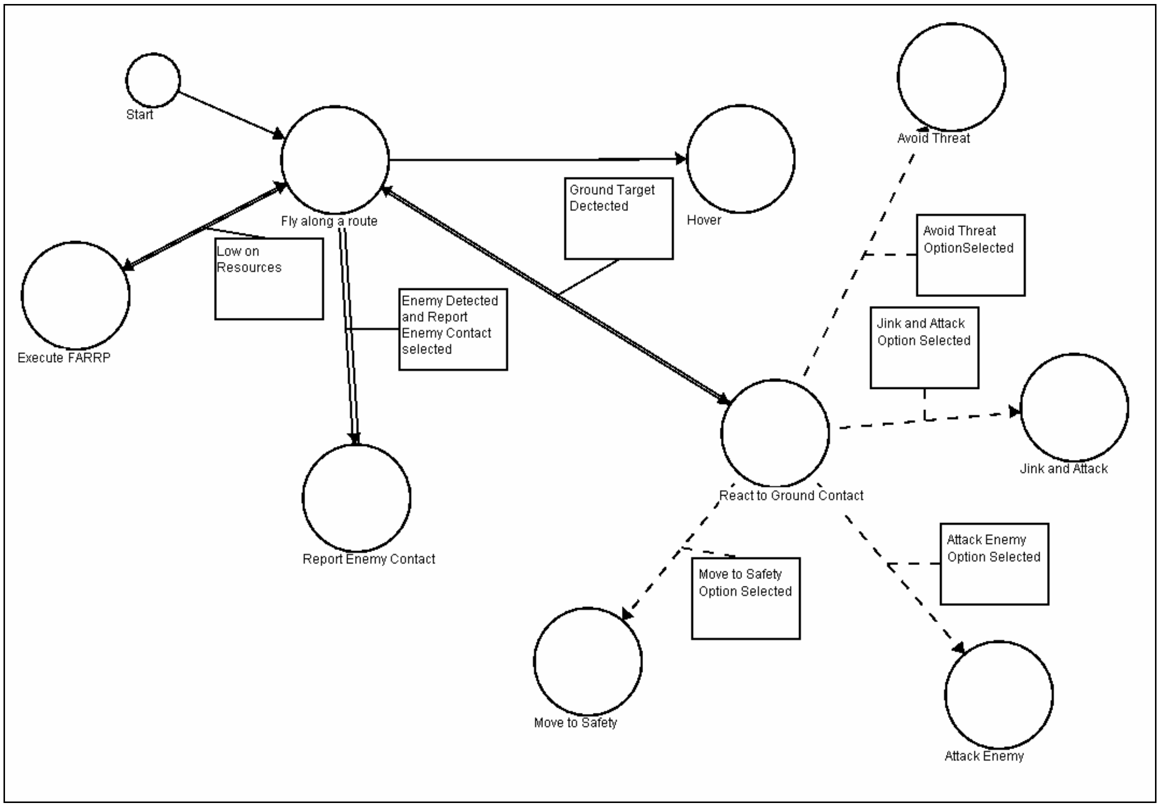


Figure 45. SBSD for RWA Fly Route Task Frame

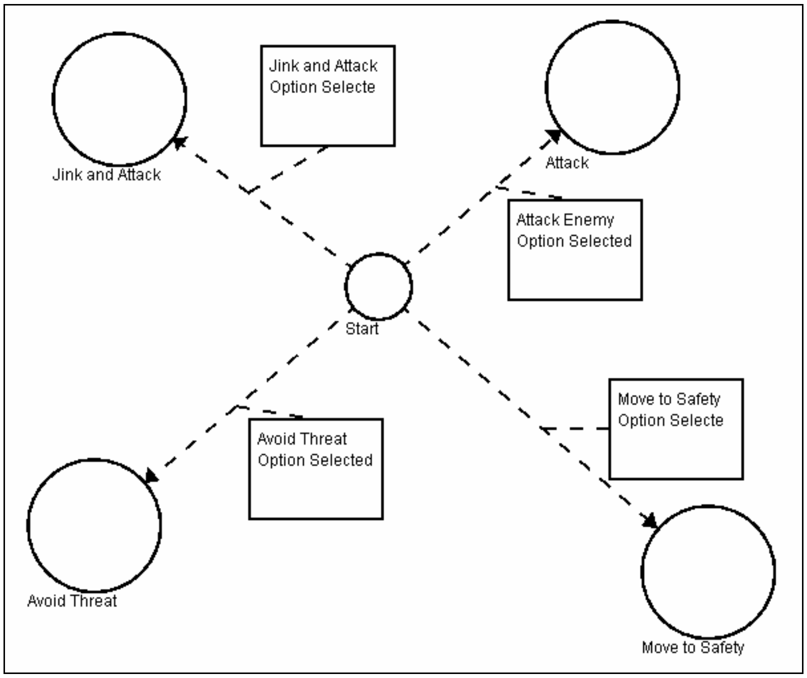

Jink and Attack
Jink and Attack Option Selecte
Attack
Attack Enemy Option Selected
Start
Move to Safety Option Selecte
Avoid Threat Option Selected
Avoid Threat
Move to Safety

Figure 46. Expanded Multi-Task node for React to Ground Contact Sub Activity

from one of several options for a sub-activity or select if an activity or reaction is turned on. Examples include either hovering or landing at the end of the FWA Ingress or how to react to spotting an enemy. In the case where the SBSD represents a task frame assigned to an entity, the diagrams will be streamlined as conditional and reactionary transitions can be removed. Figure 48 shows the SBSD diagram for the RWA fly route task assigned to one of the US CH47D vehicles in the scenario for case study three. Because the reaction to the detection of a ground target in the task frame has been defined by the creator of the scenario there is no reason to include the other reactions as possible courses of action in the diagram. The current version of SSST cannot generate the SSST diagrams for task frames assigned to entities, but could be modified to do so.

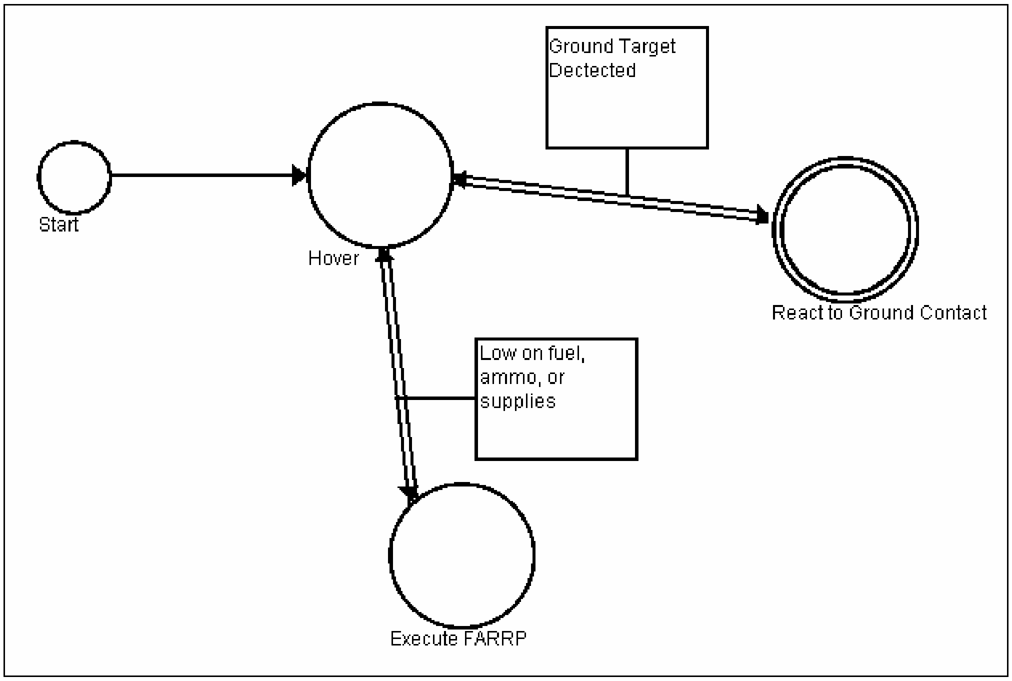


Figure 47. SBSD for Hover Task Frame

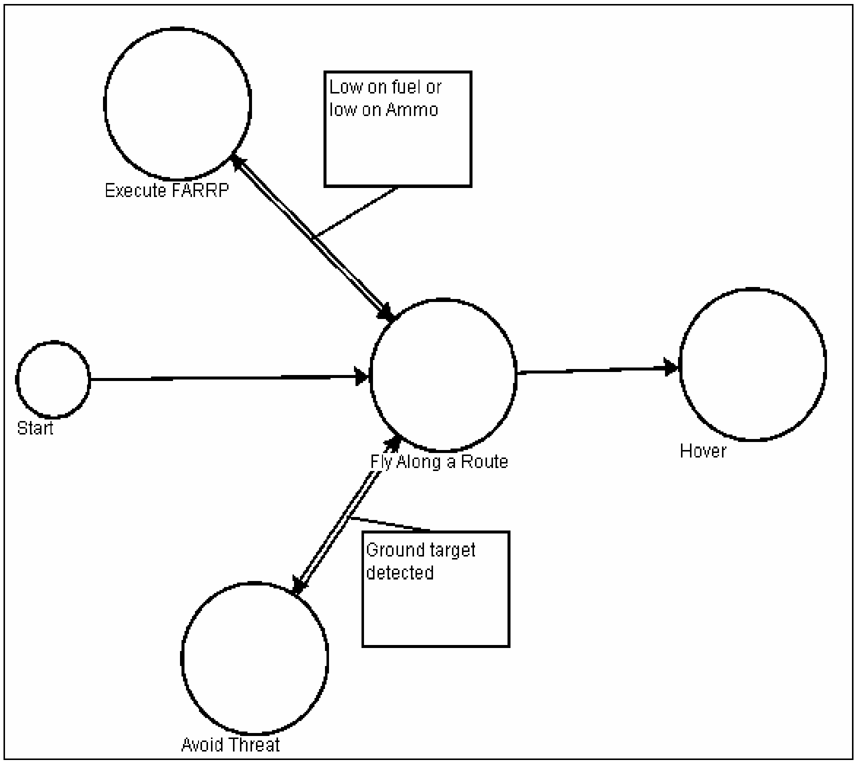


Figure 48. SBSD Diagram for the RWA Fly Route Task Frame Assigned to a US CH47D Vehicle

At the end of Case Study Two, the idea of converting task frames that had SBSD diagrams associated with them into multi-task nodes was mentioned. In Case Study Two, as each entity was only assigned one task frame, the user could easily pull up the SBSD for each task frame and see what is going on. In this case study, an entity is assigned multiple tasks, and looking at diagrams for individual task frames is no longer as desirable. Therefore, for this scenario, converting the task frames into multi-task nodes may have the potential to be more beneficial.

By converting the task frames into multi-task nodes the user can easily expand and condense the nodes to display the desired level of detail. By representing the task frames as multi-task nodes in the SBSD diagrams for the scenarios, the user can gain an overall understanding of all the behavior assigned to the entity by expanding all the nodes. However, by converting task frames into multi-task nodes, two different level of abstraction are defined for atomic nodes. Instead of just representing task frames they also represent sub-activities in the task frames. By having atomic nodes on two levels of abstraction the complexity of the language and room for errors is increased. Figure 49 shows how the SBSD diagrams would look if the task frames were converted into missions.

5.7 Evaluation of SBSD

Chapter 3 defines the evaluation criteria for SBSD. The criteria looked at the expressiveness, frequency of errors, redundancy, locality of change, reusability, reliability, translatability, and compatibility of the language. The case studies presented earlier in this chapter serve as a basis on which to evaluate the language on in the above areas.

The language proved to be expressive. All the major aspects of the simulation scenarios could be expressed. These included temporal conditions such as performing a task for a duration and the attributes assigned to the task set by the user. The language was also able to show reaction behaviors, or behaviors that were not directly assigned to the unit but performed as a response to another event. The parts of the scenario such as the reactions of the F-14 in the first case study that were not able to be expressed through the tool developed were a result of the limited capabilities implemented in the tool created and from the information not being stored in the scenarios generated by the OneSAF simulation. The language itself is sufficiently expressive.

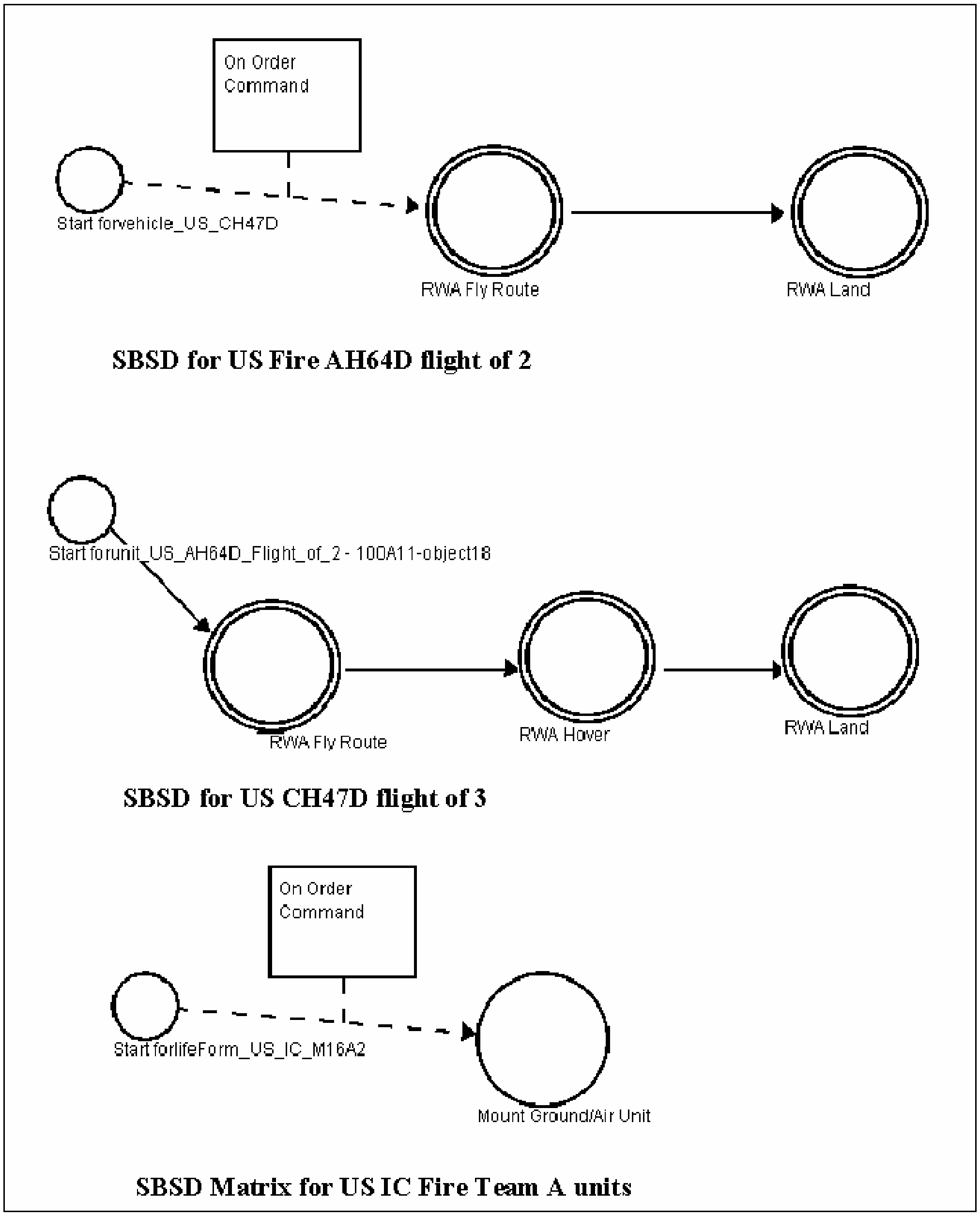

On Order Command
Start forvehicle_US_CH47D
RWA Fly Route
RWA Land
SBSD for US Fire AH64D flight of 2
Start forunit_US_AH64D_Flight_of_2 - 100A11-object18
RWA Fly Route
RWA Hover
RWA Land
SBSD for US CH47D flight of 3
On Order Command
Start forlifeForm_US_IC_M16A2
Mount Ground/Air Unit
SBSD Matrix for US IC Fire Team A units

Figure 49. SBSDs for Entities in Scenario Three with Complex TaskFrames Represented as Multi-task nodes

The one thing that was difficult to express, however, was the order of the reaction behaviors if two or more reactions took place during the execution of a behavior represented by one node. For example, in the first case study the language provides no way, outside of the attributes of the atomic node, to indicate the order in which air attack behaviors are executed.

In addition to expressiveness, the language indicates a low potential for errors. In the case studies conducted there were no errors. However the case studies are not completely representative of all the possible scenarios and were generated by a computer. If the diagrams were composed by a human the frequency of errors would most likely increase. The potential for errors is present in the misuse of the transitions. As conditional and reaction transitions diagrams are similar, it is possible they could be confused. Future work should look at human studies to verify this finding.

The case studies indicate there is no redundancy in the model. At no point in the case studies were there multiple choices for the type of node or transition to use. Each of the transitions is well-defined, so that the type of transitions represented can be broken up into distinct groups that do not overlap. The same applies to the two different types of nodes in the language. Furthermore, the automation of the diagrams from the scenario files serves as further proof of no redundancy in the language.

SBSDs also limit the impact of change in the diagram. A user can change the attributes of a node or interchange atomic nodes and missions with other atomic nodes and missions without affecting the rest of the diagram. The sequence of the nodes can also be changed, and the only nodes affected or those whose incoming or outgoing transitions are changed.

Furthermore, because the sequences of nodes can be grouped together the language also provides reusability. By defining a sequence of tasks and grouping them into a mission the user is given the ability to re-use that sequence of tasks over again.

The user is also able to change the value of the attributes of each node each time the mission is assigned, expanding the possibilities for re-use.

Translatability and reliability appear to be the weakest areas of SBSD. By itself, without the extensions of a specific simulation program, the base language is not completely reliable or translatable. However, through extensions and constraints on the base language and the development of software tools supporting the language, SBSD can have both translatability and reliability. The ability of SSST to generate SBSDs shows one aspect of the translatability. If an SBSD can be created from a scenario file, then SBSDs should also be able to be translated into the scenario language used for simulation tools. If the required parameters, for each behavior node representing a task frame in OneSAF, were incorporated into the tool developed, the diagrams would then provide the necessary information for scenario generation, when linked back to OneSAF. Due to time constraints this capability was not implemented. It is the requirement of needing constraints on the language in order to generate scenario files from SBSDs that give SBSD a low rating for translatability.

The language is also not completely reliable when applied to different simulations as SBSD diagrams are not designed to meet the constraints of the all the different simulations. However, reliability can be provided through tool support. The tools can prevent users from putting the components together in a way that is incorrect for specific simulations. The tool can also enforce required attributes of nodes and other constraints on the language. Because a tool is required to provide consistency, SBSDs do not rate high for reliability.

Finally, looking at the how the language evaluates against the above criteria one sees that it rates well in compatibility, because it evaluates well against the majority of the other criteria. There is not one area of criteria that SBSD favors strongly. Although, SBSD has reliability and translatability problems the problems are fixed by extending the language for specific simulations and tool support.

5.8 Summary

This chapter presented the application of the simulation behavior specification diagram (SBSD) and treemaps to scenarios generated by OneSAF. OneSAF is a simulation currently under testing and development by the United States Army. In OneSAF users create scenarios by assigning sequences of pre-defined tasks frames to the entities in the scenario. Task frames are a set of concurrently executing tasks and represent high-level behaviors such as “move” or “air attack.” Transitions between task frames can either take place upon the completion of a previous task or after certain temporal conditions are met. Furthermore, reactive task frames can be added to the assigned sequences of tasks in response to other events in the simulation.

In order to apply SBSD to the behaviors assigned to entities in the scenario a Java program, SSST was created. The Java program reads in a text scenario file and then creates the relationships between the entities and their assigned task in memory. After the scenario is read, users can view, edit, or create missions for the entities in the simulation. The tool has the potential then to write these modifications back out to the scenario file. The tool also has the capability to show the structure of the units in the scenarios through treemaps. Treemaps allow users to view information about scenarios that facilitates the decision-making process.

Three case studies were performed using scenarios generated by OneSAF. The case studies served as a basis for evaluating the SBSD against the criteria set in Chapter 3. The case studies demonstrated that although the language was not yet fully translatable or reliable, it was expressive, had a low frequency of errors, had no redundancy, and was re-usable. Furthermore, through the extension of SBSD to specific simulation domains SBSD will become more translatable and reliable.

VI. Conclusion and Future Work

6.1 Introduction

This chapter summarizes the research conducted in relation to the objectives stated in Chapter 1. First, the motivation behind the research is reviewed. Then the diagrams resulting from the research are discussed and evaluated. Finally, the chapter concludes with several different avenues for future work.

6.2 Motivation and Objectives

As stated in Chapter 1, the overall objective of the research conducted in this thesis was to create a visual language that aids in the comprehension and composition of composable simulation scenarios. This objective was further narrowed down to the development of a visual language that describes the behavior of components serving as entities in simulation scenarios and to the application of treemaps to the hierarchy of entities in the scenarios.

The use of a visual language to describe simulation scenarios makes scenarios easier to comprehend and build for several reasons. First, diagrams are more useful than text for these purposes because they help the learner build mental models that demonstrate how processes work. Diagrams are also better at showing relationships than text alone.

However, despite all the current behavior specification methods currently in use, none of the specifications were suited for the representation of the high-level behavior of entities in simulations. In particular, none of the diagrams allowed for the distinction between behaviors that were assigned and behaviors that occurred as a result of events in the simulation. Therefore Simulation Behavior Specification Diagrams (SBSDs) were developed.

6.3 Simulation Behavior Specification Diagrams

SBSDs are a variation of process dependency diagrams designed to represent the high-level behavior of entities in battlefield simulations. The components of the diagrams are atomic nodes, multi-task nodes, regular transitions, conditional transitions, permanent reaction transitions, and temporary reaction transitions. The nodes represent behaviors executed by the entity they are assigned to while the transitions specify the end of an entity performing one behavior and starting another behavior.

SBSDs directly support composability by allowing for a sequence of tasks to be grouped together into a multi-task node. The multi-task node can then be used in other sequences of behaviors assigned to units and entities or other multi-task nodes. By providing this capability, the diagrams allow the user to specify the level of detail desired and promote re-use of commonly used sequences of behaviors. Like atomic nodes, the multi-task nodes have attributes that can be modified by the user.

SBSDs also support the idea of reaction transitions. In a mission-level model, an entity is assigned a mission or sequence of tasks. However during the execution of the mission the entity might divert to perform another task based on what occurs in the environment of the simulation. SBSDs allow for these behaviors to be marked by a special type of reaction. Furthermore, whether or not the entity returns to the original behavior is also indicated.

Upon evaluation of the language, SBSD was able to represent all of the aspects needed to accurately represent the target domain. Through the inclusion of reaction transitions the diagrams are able to show reactions, while through conditional transitions temporal conditions are able to be represented. Composability and a high-level of abstraction are provided by the inclusion of multi-task nodes which allow the user to control the level of abstraction shown in the diagrams. Furthermore, through the use of attributes in the nodes, parameters specified by the user in scenario development are also represented in SBSDs.

In order to evaluate the language a Java program, Simulation Scenario Specification Tool (SSST), was implemented to apply SBSDs to OneSAF simulation scenarios. OneSAF is a mission-level simulation used by the United States Army. SSST works by reading in the simulation scenarios created by OneSAF and then displaying the behaviors either assigned to, or performed by, the entities in the simulation. Through the use of SSST, three case studies were conducted on OneSAF simulation scenarios.

Examination of SBSDs using the evaluation criteria stated in Chapter 3 showed the language rates highly in the areas of expressiveness, frequency of errors, redundancy, locality of change, and reusability. The two areas the diagrams rated low in were translatability and reliability. By extending and constraining the language for specific simulations the language will rate higher in translatability and reliability.

6.4 Application of Treemaps

In addition to the creation of SBSD, the information visualization technique of treemaps was also applied the hierarchy of units in mission-level scenarios. Treemaps are an information visualization technique used to aid in decision-making. Several different aspects of the scenarios were able to be shown through treemaps. In the treemap portion of SSST, the size of the forces and the capability of the forces were visually shown using treemaps. Treemaps can visually show users the size of forces and entities in forces without having to textually display the size of each entity. Treemaps also provide the user a view of all the entities participating in the scenario without having to look at multiple pages or expand and minimize nodes in a tree.

6.5 Future Work

Further research that can be conducted in the area of visual representations for composable simulations can be divided into two main areas. The first main area is the further development and application of the Simulation Behavior Specification Diagram (SBSD), while the second area focuses on adapting UML or creating new visual languages to represent the aspects of composable simulations that are not addressed by SBSD or treemaps.

6.5.1 Further Development of SBSD

The research conducted for this thesis addresses only the basics of the high-level behavior of entities in simulation scenarios. It is designed to be extendable to describe the behavior of entities in multiple levels of simulation. In order to make SBSD usable for multiple simulations more work needs to be conducted on the application of SBSD to other simulation systems outside of OneSAF. Ideally, a tool that can read in several different simulation scenario files and then allow the user to view and edit the behavior of the entities in the system can be created. By having a tool that can read multiple simulation scenarios, the next step of converting scenarios for one simulation into another simulation is closer.

More work needs to be done on the program used to apply SBSD to OneSAF. Although the current tool built to apply SBSD to OneSAF allows for the behavior of entities in the scenarios to be viewed, the user cannot modify the behavior and then save the scenario back out to a file. Other current shortcomings of the tool are that the capacities of users to view and edit the parameters of the behaviors assigned to entities is limited to a select few tasks. In order for SBSD to be completely integrated with OneSAF, the user will need to be given the ability to have the same functionality given to them by the OneSAF GUI in the tool that applies SBSD to the simulation scenarios.

Finally, another direction of research is the creation of a tool that allows the user to adapt the components of SBSD to meet the syntax and semantics of the simulation they are currently working on. By creating a tool that allows users to define their own syntax for the components, it gives the user more control and flexibility in the use of SBSD.

6.5.2 Expansion of the Visual Language for Simulation Scenarios

Another area that further research can be conducted in is the expansion of either UML or the creation of a new visual language to describe aspects of composable simulations and simulation scenarios outside of behavior specification. The research conducted for this thesis did not look at several aspects of composable simulations that can be described visually. Specifically the research did not address the architecture or definition of the entities used in the simulation. Nor do the diagrams presented express communications or relationships between the entities outside of the hierarchical command structure of the entities. Finally the diagrams presented do not address the structure of the simulation systems. These areas are needed in a visual language that completely represents simulation scenarios. Two options in the development of this language are to create either a new visual language or adapt a current visual language such as UML. UML is a potential candidate language as it already addresses some of the aspects addressed above. The visual meta-language for generic modeling language created by Hakan Canli is also another potential language that can be expanded to describe the different aspects of simulation scenarios [CAN02].

6.6 Summary

In the world of simulation and modeling, composable simulations have the potential to offer many benefits; however, many obstacles need to be overcome first. In this research, a visual language is applied to simulation and modeling in order to reduce the complexity of, and serve as a standard descriptor for, certain aspects of composable simulation scenarios. The results of this research are Simulation Behavior Specification Diagrams (SBSD) and the application of treemaps to simulation scenarios. SBSDs proved to be better suited to represent the high-level behavior of entities in simulation scenarios then the other behavior specification techniques studied. Case studies showed that SBSDs were expressive, had a low frequency of errors, had no redundancy, and support reusability. By placing constraints on the language and through tools the translatability and reliability of the language can be increased. Furthermore, treemaps were successfully applied to the hierarchy of entities in scenarios to visually display the size of each unit in the hierarchy. The user can also define other properties of the treemap, such as color, in order to visually display additional properties of the hierarchy and units in the hierarchy.

Bibliography

[ALL00] Allain, Laurent and P. Yim. "Modeling Information System Behavior with Dynamic Relations Nets," *Journal of Universal Computer Science,* 10:1109-1130 (November 2000).

[ARO99] Aronson, Jesse and P. Bose. "A Model-Based Approach to Simulation Composition," *Proceedings of the 1999 Symposium on Software Reusability.* 78-82. Los Angeles: 1999.

[BID00] Biddle, Mark and P. Constance. "An Architecture for Composable Interoperability," *Proceedings of the 2000 Spring Simulation Interoperability Workshop.* 2000

[BUS03] Bush, Frank. OneSAF TEMPO Representative. Electronic Message. 13 January 2003.

[BRO00] Brook, Phillip, J. Ostroff, and R. Paige. "Principles for Modeling Language Design," *Information and Software Technology*, 42: 665-675 (July 2000).

[CAN02] Canli, Hakan. *A Visual Meta-Language For Generic Modeling.* MS thesis, AFIT/GCE/ENG/02M-1. Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2002.

[CHA99] Chang, Xinjie. "Network Simulations with Opnet," *Proceedings of the 31st Conference on Winter Simulation.* 307-314. Phoenix: 1999.

[CLA99] Clark, Tony, A. Evans, R. France S. Kent, Stuart, and B. Rumpe. "Response to UML 2.0 Request for Information Submitted by the precise UML group." Document submitted in response to the OMG's request for information. n. pag. http://cgi.omg.org/docs/ad/99-12-16.pdf. July 2002.

[COU97] Courtemanche, Anthony, S. von der Lippe, and J. McCormick. "Developing User-Composable Behaviors," *Proceedings of the 1997 Fall Simulation Interoperability Workshop.* Orlando, FL: Institute for Simulation and Training, 1997.

[CRA98] Crain, Robert. "Simulation Using GPSS/H," *Proceedings of 29 Conference on Winter Simulation*. 567-573. Atlanta: ACM Press,1997

[DAH98] Dahmann, Judith, R.Fujimoto, and R.Weatherly. “The DoD High-level Architecture: An Update,” *Proceedings of the 30th Confernce on Winter Simulation.* 797-804. Washington D.C.: IEEE Computer Society Press,1998

[DAV00] Davis, Paul, P. Fishwick, M.Overstreet, and D. Pegden. “Model Composability as a Research Investment: Responses to the Featured Paper,” *Proceedings of the 32nd Conference on Winter Simulation.* 1585-1591. Orlando: Society for Computer Simulation International, 2000

[DES00] Desel, Jörg. “Teaching System Modeling, Simulation, and Validation” *Proceedings of the 32nd Conference on Winter Simulation.* 1669-1675. Orlando: Society for Computer Simulation International, 2000

[DMS02] Defense Modeling and Simulation Office. “Proposal For Composable Modeling and Simulation Studies.” 2002.

[DMS02b] Defense Modeling and Simulation Office. “Concept of Operations for Composable Modeling and Simulation,” *Workshop on Composable Modeling and Simulation.* 2002.

[DMS02c] Defense Modeling and Simulation Office. “Notional Composable Modeling and Simulation Languages, Databases, and Data,” *Workshop on Composable Modeling and Simulation.* 2002.

[DMS02d] Defense Modeling and Simulation Office. “Component-based Architecture and Modeling and Simulation,” *Workshop on Composable Modeling and Simulation.* 2002.

[DMS02e] Defense Modeling and Simulation Office. “Two Aspects of Composability:Lexicon and Theory,” *Workshop on Composable Modeling and Simulation.* 2002.

[DMS95] Directorate of Modeling, Simulation, & Analysis. *Modeling and Simulation Master Plan.* Web Document. pag. n. http://www.afams.af.mil/webdocs/afmsmp/. August 2002.

[FLO02] Flower, Martin and C. Kobryn. “Customizing UML for Fun and Profit.” Article from Software Development Online. pag. n. http://www.sdmagazine.com/documents/s=7224/sdm0207d/0207d.htm. July 2002.

[FLO99] Flower, Martin and K.Scott. *UML Distilled* (2nd edition). Boston: Addison-Wesley Publishing Company,1999.

[HOR98] Horn, Robert. *Visual Language Global Communication for the 21st Century.* MacRovu Inc, 1999.

[LAR01] Larsen, Karin, C.Burns, O. Nnedu, and C. Sons. "Entity and Behavior Composition in CGF Simulations Using the Synthetic Common Operating Environment," *Proceedings of the 1997 Fall Simulation Interoperability Workshop.* 2001.

[JO9H1] Johnson, Brian and B. Shneiderman. "TreeMaps: A Space-filling Approach to Visualization of Hierarchical Information Structures",*Proc. of the 2nd International IEEE Visualization Conference.* 284-291. San Diego: Oct. 1991.

[KAS00] Kasputis, Stephen and H. Ng. "Model Composability: Formulating a Research Thrust: Composable Simulations", *Proceedings of the 32nd Conference on Winter Simulation.* 1577-1584. Orlando: Society for Computer Simulation International, 2000.

[MAR01] Martinez, Julio. "EZStrobe- General-Purpose Simulation System Based on Activity Cycle Diagrams," *Proceedings of the 33nd Conference on Winter Simulation.* 1556 – 1564. Arlington: IEEE Computer Society, 2001.

[MCC00] McCoramck, Jenifer and S. von der Lippe, "Embracing Temporal Relations in Composable Behaviors Technology." *Proceedings of the 1999 Fall Simulation Interoperability Workshop.* 1999.

[OMG01] Object Management Group. "Unified Modeling Language (UML), Version 1.4", Specification for the Unified Modeling Language. n. pag. http://www.omg.org/technology/documents/formal/uml.htm. Published 2001, accessed July 2002.

[OMG02] Object Management Group. "Introduction to OMG's Unified Modeling Language (UML)," Unpublished article. n. pag. http://www.omg.org/gettingstarted/what_is_uml.htm. July 2002.

[PUC01] Puckett, Jason, K. Johnson, and B. Wise. "Using Finite State Machines For Behavior Representation With Composable Objects in NASM," *Proceedings of the 1999 Spring Simulation Interoperability Workshop.* 1999.

[RIC00] Richter, Hendrick and L. Marz. "Toward a Standard Process: The Use of UML for Designing Simulation Models," *Proceedings of the 32nd Conference on Winter Simulation.* 394-398. Orlando: Society for Computer Simulation International, 2000.

[SEL94] Selic, Brian, G. Gullekson, and P. Ward. *Real Time Object Oriented Modeling.* John Wiley & Sons, 1994.

[STY01] Stytz, Martin and S. Banks. "Enhancing the Design and Documentation of High-level Architecture Simulations Using the Unified Modeling Language," *Proceedings of the 1999 Spring Simulation Interoperability Workshop.* 1999.

[STR03A] Stricom. "History of OneSaf," Unpublished article. n. pag. http://www.onesaf.org/public1saf.html. September 2002.

[STR03B] Stricom. "OTB 1.0 Software Architecture Design and Overview Document," OneSAF CD. November 2000

[STR03C] Stricom. "OTB 1.0 Users Manual Vol. 1," OneSAF CD. November 2000.

[STR03D] Stricom. "OTB 1.0 Users Manual Vol. 2, " OneSAF CD. January 1999.

[VON99] von der Lippe, Sonia, J. McCormick, and M. Kalphat. "Embracing Temporal Relations and Command and Control in Composable Behaviors Technology," *Proceedings of the 9th Conference on Computer Generated Forces and Behavioral Representation.* 1999

[WIE98] Wieringa, Roel. "A Survey of Structured and Object-Oriented Software Specification Methods and Techniques," *ACM Computing Surveys*, 30: 459-527 (December 1998).

[ZIM02] Zimmerman, Armin. "Petri nets," Unpublished article. n. pag. http://pdv.cs.tu-berlin.de/~azi/petri.html. September 2002

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 074-0188*

1. REPORT DATE *(DD-MM-YYYY)* 25-03-2003	2. REPORT TYPE **Master's Thesis**	3. DATES COVERED *(From – To)* Jun 2002 – Mar 2003

4. TITLE AND SUBTITLE A VISUAL LANGUAGE FOR COMPOSABLE SIMULATION SCENARIOS	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Bartley, Carolyn, R., 2nd Lieutenant, USAF	5d. PROJECT NUMBER 2002-079
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way, Building 640 WPAFB OH 45433-7765	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/03-03
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Philomena M. Zimmerman Defense Modeling and Simulation Office 1901 North Beauregard Street, Suite 500 Alexandria, VA 22311-1705 (703) 998-0660 email: pzimmerman@dmso.mil	10. SPONSOR/MONITOR'S ACRONYM(S) DMSO
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Modeling and Simulation plays an important role in how the Air Force trains and fights. Scenarios are used in simulation to give users the ability to specify entities and behaviors that should be simulated by a model: however, building and understanding scenarios can be a difficult and time-consuming process. Furthermore, as composable simulations become more prominent, the need for a common descriptor for simulation scenarios has become evident.

In order to reduce the complexity of creating and understanding simulation scenarios, a visual language was created. The research on visual languages presented in this thesis examines methods of visually specifying the high-level behavior of entities in scenarios and how to represent the hierarchy of the entities in scenarios. Through a study of current behavior specification techniques and the properties of mission-level simulation scenarios, Simulation Behavior Specification Diagrams (SBSD) were developed to represent the behavior of entities in scenarios. Additionally, the information visualization technique of treemaps was adapted to represent the hierarchy of entities in scenarios.

After completing case studies on scenarios for the OneSAF simulation model, SBSDs and the application of treemaps to scenarios was considered successful. SBSD diagrams accurately represented the behavior of entities in the simulation scenarios and through software can be converted into code for use by simulation models. The treemap displayed the hierarchy of the entities along with information about the relative size of the entities when applied to simulation scenarios.

15. SUBJECT TERMS

Simulation, Combat Simulation, Simulation Languages, Model

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES 163	19a. NAME OF RESPONSIBLE PERSON Lt Col Karl S. Mathias
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	UU		19b. TELEPHONE NUMBER *(Include area code)* (937) 255-3636, ext 4716; e-mail: Karl.Mathias@afit.edu